Titre: Title:	Architectures matérielles et bibliothèques personnalisées pour la simulation en temps réel sur FPGA
Auteur: Author:	Fahimeh Hajizadeh
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Hajizadeh, F. (2025). Architectures matérielles et bibliothèques personnalisées pour la simulation en temps réel sur FPGA [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/68126/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Architectures matérielles et bibliothèques personnalisées pour la simulation en temps réel sur FPGA

FAHIMEH HAJIZADEH
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Août 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Architectures matérielles et bibliothèques personnalisées pour la simulation en temps réel sur FPGA

présentée par **Fahimeh HAJIZADEH**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Guy BOIS, président
Jean Pierre DAVID, membre et directeur de recherche
Tarek OULD-BACHIR, membre et codirecteur de recherche
Yvon SAVARIA, membre
Lahoucine ID-KHAJINE, membre externe

DÉDICACE

To coffee, late nights, and that one MATLAB bug

that almost earned a co-authorship.

REMERCIEMENTS

Cette thèse de doctorat représente pour moi l'aboutissement d'un long parcours académique, qui n'aurait pas été possible sans le soutien de toutes les personnes qui m'ont accompagnée tout au long de ce chemin.

Tout d'abord, je tiens à remercier mon directeur, le Professeur Jean Pierre David, pour la confiance qu'il m'a accordée et pour son accompagnement tout au long de cette aventure. Je lui suis particulièrement reconnaissante de m'avoir transmis non seulement des connaissances académiques, mais aussi des leçons de vie précieuses, qui ont eu un impact durable sur moi. Je suis également profondément reconnaissante envers mon co-directeur, le Professeur Tarek Ould-Bachir, pour son soutien indéfectible, ses idées éclairées et ses discussions stimulantes. Son temps et ses commentaires précieux m'ont continuellement poussée à m'améliorer. Travailler sous sa supervision a été à la fois un honneur et une source d'inspiration.

J'exprime aussi ma plus profonde gratitude à mon mari, Mostafa Abbasmollaei, pour son amour inconditionnel, son soutien constant, et pour avoir été mon pilier au quotidien. Sa patience, son encouragement et sa confiance inébranlable en moi m'ont portée dans les moments les plus difficiles. Il a toujours été mon refuge, mon plus grand soutien, et celui qui me rappelait ma force lorsque je doutais de moi. Ce parcours n'aurait pas été possible sans lui à mes côtés.

À ma famille élargie — mon père, mon héros pour toujours, dont la force et l'intégrité m'ont toujours inspirée ; ma mère, qui a supporté l'éloignement avec grâce et m'a toujours soutenue sans condition ; ma sœur et mes frères, qui ont toujours été là pour moi — je vous remercie pour votre amour, votre encouragement et votre présence dans ma vie.

Enfin, je remercie les membres du jury d'avoir pris le temps de lire, d'évaluer et de fournir des commentaires précieux sur ce travail.

RÉSUMÉ

Cette thèse propose une méthodologie structurée et de haut niveau pour l'implémentation des équivalents de réseau dépendant de la fréquence (Frequency-Dependent Network Equivalent, FDNE) sur des circuits logiques programmables (Field-Programmable Gate Array, FPGA), en vue de la simulation en temps réel des systèmes électriques. Les simulateurs en temps réel permettent d'évaluer les spécifications, de vérifier le comportement des systèmes modélisés au cours des différentes phases de développement, et de faciliter leur intégration, notamment dans le cadre des configurations Hardware-in-the-Loop (HIL). Ils contribuent à réduire les risques et les coûts liés à l'utilisation de bancs d'essai physiques. Toutefois, leur implémentation sur FPGA demeure une tâche complexe, traditionnellement réservée aux experts, en raison de cycles de développement longs et de la complexité du design matériel bas niveau.

Pour répondre à ces défis, ce travail s'appuie sur des outils de synthèse de haut niveau (High-Level Synthesis, HLS) tels que Vivado HLS de Xilinx, qui permettent de générer du matériel à partir de code écrit en C/C++/SystemC, tout en offrant la possibilité d'orienter la synthèse via des directives. Bien que cette approche accélère le développement et la vérification fonctionnelle, les performances atteintes peuvent rester inférieures à celles des conceptions RTL optimisées manuellement, notamment en raison du manque de mécanismes efficaces pour instancier des modules personnalisés.

Dans ce contexte, cette recherche propose plusieurs approches de modélisation des FDNEs fondées sur la représentation dans l'espace d'état, mises en œuvre sur des plateformes FPGA. L'étude examine également l'impact de la résolution numérique à travers l'utilisation de différents formats de données : virgule flottante standard (simple et double précision), virgule fixe et virgule flottante personnalisée. Deux bibliothèques HLS dédiées, CuFP et CuFPSAF, ont été développées pour permettre des opérations arithmétiques personnalisées et une gestion efficace de l'alignement des exposants. La bibliothèque CuFPSAF a démontré des gains notables en performance, avec une réduction de latence de 29,4 % par rapport à CuFP, et de plus de 80 % par rapport aux cœurs IP commerciaux, tout en réduisant l'utilisation de DSP de 59,7 %. Par ailleurs, une architecture d'additionneur optimisée, nommée IESC-HRCS, a été introduite pour les opérations intensives en addition. La thèse propose également un cadre de simulation en temps réel basé sur FPGA pour les convertisseurs connectés au réseau, visant à renforcer la stabilité du réseau électrique par une modélisation précise des dynamiques dépendantes de la fréquence à l'aide de modèles FDNE. Une validation effectuée à l'aide d'un compensateur synchrone statique (STATCOM) connecté à un réseau de 500 kV a confirmé

la précision du modèle, avec une erreur relative 2-norme aussi faible que 0,0107 % pour la configuration CuFP (11, 53). Le cadre proposé atteint des latences inférieures à une microseconde, atteignant notamment 312 ns avec CuFP (8, 24), démontrant ainsi son aptitude à la simulation de transitoires électromagnétiques (Electromagnetic transients, EMT) à haute résolution.

ABSTRACT

This thesis proposes a structured and high-level methodology for implementing Frequency-Dependent Network Equivalents (FDNEs) on Field-Programmable Gate Arrays (FPGAs) for the real-time simulation of power systems. Real-time simulators enable the evaluation of system specifications, verification of system behavior throughout development phases, and facilitate system integration, particularly in Hardware-in-the-Loop (HIL) environments. They significantly reduce the risks and costs associated with physical test benches. However, implementing real-time simulators on FPGAs remains a complex task traditionally reserved for experts, due to long development cycles and the intricacies of low-level hardware design.

To address these challenges, this research leverages High-Level Synthesis (HLS) tools such as Xilinx Vivado HLS, which enable the generation of hardware from C/C++/SystemC code and support design guidance through synthesis directives. While HLS accelerates development and functional verification, its generated designs may not match the efficiency of manually optimized RTL implementations, and it lacks support for effectively instantiating custom hardware modules.

To overcome these limitations, this work presents several modeling approaches for FDNEs based on state-space representation, implemented on FPGA platforms. It further investigates the impact of numerical resolution on simulation accuracy and performance, using a variety of data types including standard floating-point (single and double precision), fixed-point, and customized floating-point formats. Two dedicated HLS libraries—CuFP and CuFPSAF—were developed to support customizable floating-point arithmetic and efficient exponent alignment. CuFPSAF demonstrated significant performance gains, reducing latency by 29.4% compared to CuFP and by over 80% compared to vendor IP cores in vector summation, while also reducing DSP usage by 59.7%. An improved adder architecture, IESC-HRCS, was also introduced to optimize addition-heavy operations. Furthermore, the thesis proposes an FPGA-based real-time simulation framework for grid-tied converters, with a focus on enhancing grid stability through accurate modeling of frequency-dependent network dynamics using FDNEs. Validation using a Static Synchronous Compensator (STATCOM) connected to a 500 kV network showed strong correlation with reference models from EMTP®, achieving a 2-norm relative error as low as 0.0107% with CuFP (11, 53). The framework demonstrated sub-microsecond latencies, reaching values as low as 312 ns with CuFP (8, 24), confirming its suitability for high-resolution EMT simulation.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BE	Backward Euler : schéma d'intégration implicite d'Euler utilisé en simulation numérique
CLA	Carry Look-ahead Adder : anticipateur de retenue
CPU	Central Processing Unit : acronyme anglais servant à désigner un PUG, et par métonymie un ordinateur de bureau
CuFP	Customized Floating-Point : format flottant personnalisé
CuFPSAF	Customized Floating-Point with Self-Alignment Format : format flottant personnalisé avec alignement automatique
DER	Distributed Energy Resource : ressource énergétique distribuée
DFP	Double-Precision Floating-Point : virgule flottante double précision
DP	Dot Product (operator) : produit scalaire, acronyme utilisé dans nos publications anglophones
DRTS	Digital Real-Time Simulation : simulation numérique en temps réel
DSP	Digital Signal Processing : traitement de signal numérique
EDC-HRCS	Endomorphic Double Carry-Chain HRCS : additionneur HRCS endomorphique à double chaîne de retenue
EMT	Electromagnetic Transient : méthode numérique de modélisation des réseaux électriques employant l'analyse nodale
EMTP	Electromagnetic Transient Program : logiciel de simulation des réseaux électriques de type EMT
ESC-HRCS	Endomorphic Single Carry-Chain HRCS : additionneur HRCS endomorphique à chaîne de retenue simple
FDNE	Frequency-Dependent Network Equivalent : équivalent de réseau dépendant de la fréquence
FF	Flip-Flop : bascule
FloPoCo	FLoating Point Operator COre : générateur libre de cœurs arithmétiques paramétrables
FP	Floating-Point : virgule flottante
FPGA	Field-Programmable Gate Array : circuit numérique fait de réseaux pré-diffusés programmables
FTRT	Faster-Than-Real-Time : plus rapide que le temps réel
FXP	Fixed-Point : virgule fixe
HDL	Hardware Description Language : langage de description matériel

HIL	Hardware-In-the-Loop : configuration de simulation en temps réel avec boucle matérielle
HLS	High-Level Synthesis : synthèse de haut niveau
HRCS	High-Radix Carry-Save : additionneur à sauvegarde de retenues à base élevée
IESC-HRCS	Improved Endomorphic Single Carry-Chain HRCS : version améliorée de l'additionneur HRCS à chaîne simple
IP	Intellectual Property (core) : cœur de propriété intellectuelle
LC	Latency-Constrained : contraint en latence
LUT	Look-Up Table : table de correspondance
MANA	Modified-Augmented Nodal Analysis : analyse nodale modifiée et augmentée
MNA	Modified Nodal Analysis : analyse nodale modifiée
MVM	Matrix-Vector Multiplication : multiplication matrice-vecteur, acronyme utilisé dans nos publications anglophones
OOP	Object-Oriented Programming : programmation orientée objet
RC	Resource-Constrained : contraint en ressources
RCA	Ripple Carry Adder : additionneur à propagation de retenue
RTL	Register-Transfer Level : niveau transfert de registre
SAF	Self-Aligned Format : format auto-aligné
SAT	Self-Alignment Technique : technique d'auto-alignement
SFP	Single-Precision Floating-Point : virgule flottante simple précision
SPS	SimPower Systems : bibliothèque Simulink pour la modélisation des réseaux électriques
STATCOM	Static Synchronous Compensator : compensateur synchrone statique
THLS	Template HLS : synthèse de haut niveau basée sur des modèles C++
TSA	Transient Stability Analysis : analyse de la stabilité transitoire
VSC	Voltage Source Converter : convertisseur à source de tension
VSUM	Vector Summation : sommation vectorielle, acronyme utilisé dans nos publications anglophones

CHAPITRE 1 INTRODUCTION

1.1 Context

La complexité croissante des systèmes électriques modernes—incluant la production, le transport et la distribution d'électricité, ainsi que les véhicules électriques et les applications aérospatiales—nécessite des phases rigoureuses de test et de validation à travers l'ensemble du cycle de développement. Depuis plusieurs décennies, la simulation s'impose comme un outil efficace, tant dans le milieu industriel qu'académique, offrant un environnement rentable et sans risque pour évaluer le comportement d'un système avant sa mise en œuvre physique. Le système physique est modélisé à l'aide d'un ensemble d'équations décrivant son comportement dans la plage de fonctionnement visée. La puissance de calcul et le temps requis pour exécuter les calculs associés dépendent de la complexité du modèle.

Les processus de simulation sont généralement classés en deux catégories : simulation hors ligne et simulation en temps réel. Dans une simulation hors ligne, les équations du système sont résolues sans contrainte temporelle, permettant l'obtention de son état et de ses sorties indépendamment du temps nécessaire à l'exécution des calculs. Ce type de simulation offre une grande flexibilité et est généralement utilisé dans les premières étapes du développement. Toutefois, l'absence de synchronisation temporelle rend la simulation hors ligne inadaptée aux applications nécessitant une réponse dans des délais stricts. À l'inverse, la simulation en temps réel réalise tous les calculs du modèle dans la durée d'un pas de temps fixe et produit des sorties de manière synchronisée, respectant des contraintes strictes de temps réel.

La simulation en temps réel facilite les tests et la vérification de systèmes réels au cours des différentes phases du processus de développement. Elle permet de raccourcir les cycles d'itération, réduisant ainsi les coûts et les délais d'implémentation. Pour cette raison, la simulation en temps réel constitue une alternative privilégiée au prototypage matériel dans le développement de dispositifs et de systèmes. Elle permet en outre de tester un système en cours de développement dans des conditions extrêmes, sans exposer les équipements physiques à des risques. Les simulateurs temps réel sont couramment utilisés pour tester des contrôleurs connectés à un modèle virtuel du système physique selon une configuration de type hardware-in-the-loop (HIL). Dans ce cadre, le contrôleur réel est relié au simulateur et ne perçoit aucune différence entre le modèle simulé et le système réel.

Historiquement, les simulateurs en temps réel étaient basés sur des plateformes utilisant des processeurs centraux (CPU) ou des grappes de serveurs (PC clusters), pour répondre aux

exigences de performance. Cependant, dans les applications d'électronique de puissance où les fréquences de commutation dépassent souvent plusieurs dizaines de kilohertz, il devient nécessaire d'atteindre des pas de temps de simulation de l'ordre de la sous-microseconde pour reproduire fidèlement les phénomènes dynamiques rapides. Les contraintes liées à la nature séquentielle et aux temps de communication des architectures CPU rendent difficile l'atteinte de ces performances. En conséquence, les circuits logiques programmables de type FPGA (Field-Programmable Gate Array) sont apparus comme une solution alternative prometteuse, grâce à leur parallélisme natif et leur exécution déterministe.

Malgré leurs avantages en termes de performance, les simulateurs en temps réel basés sur FPGA ont été traditionnellement développés à l'aide de méthodologies de conception bas niveau selon un flot de type Register-Transfer Level (RTL), utilisant des langages de description matérielle (HDL) comme le VHDL ou le Verilog. Bien que puissants, ces outils requièrent un savoir-faire spécialisé, et le processus de développement est généralement long et complexe. Une simple modification peut nécessiter plusieurs jours, voire semaines, ce qui ralentit considérablement la productivité et retarde la mise sur le marché.

Pour pallier ces limitations, les outils de synthèse haut niveau (High-Level Synthesis, HLS) ont gagné en popularité. Ces outils permettent de concevoir des architectures matérielles en utilisant des langages de programmation de haut niveau comme le C ou le C++, lesquels sont ensuite compilés en code RTL synthétisable. Cette approche offre un gain de productivité significatif, rendant le développement FPGA plus accessible aux ingénieurs applicatifs et aux développeurs logiciels. Toutefois, un défi majeur persiste : la qualité des résultats générés par les outils HLS reste inférieure à celle des conceptions RTL optimisées manuellement, en particulier pour les applications de simulation en temps réel où les contraintes de latence, de débit et d'utilisation des ressources sont cruciales.

L'obtention de pas de temps de simulation inférieurs à une microseconde à l'aide d'architectures générées par HLS sur FPGA demeure un défi de recherche de premier plan. Cette thèse s'inscrit dans cette problématique en proposant une méthodologie structurée reposant sur l'utilisation de HLS, de formats numériques personnalisés et de bibliothèques arithmétiques optimisées, afin de concevoir des simulateurs en temps réel efficaces et évolutifs sur FPGA, avec un accent particulier sur la modélisation et la simulation des modèles de réseaux équivalents à dépendance fréquentielle (Frequency-Dependent Network Equivalents, FDNE).

1.2 Définition du problème

La demande en simulation temps réel des systèmes électriques a fortement augmenté ces dernières années, en raison de la complexité croissante des réseaux électriques modernes et du déploiement généralisé des convertisseurs électroniques de puissance dans les réseaux intelligents, les systèmes d'énergie renouvelable et les transports électrifiés. Les plateformes industrielles de simulation ont évolué vers des architectures hétérogènes combinant des processeurs et des circuits logiques programmables. Dans les applications nécessitant des temps de calcul extrêmement courts, les circuits logiques programmables sont souvent utilisés pour compléter ou remplacer les processeurs, qui ne peuvent à eux seuls satisfaire les contraintes temporelles strictes imposées par la simulation temps réel. Les simulateurs temps réel basés sur FPGA doivent offrir un haut débit de calcul, une grande flexibilité architecturale, des mécanismes d'interconnexion efficaces et une latence de réponse extrêmement faible. Ces exigences deviennent particulièrement critiques dans la simulation des phénomènes électromagnétiques transitoires, où les pas de temps doivent atteindre des résolutions submicrosecondes afin de capturer les fréquences de commutation élevées. Grâce à leur parallélisme intrinsèque et à leur configurabilité, les FPGAs se prêtent bien au développement de solveurs matériels à haute vitesse adaptés à la simulation temps réel de réseaux électriques complexes.

Par ailleurs, l'augmentation de l'échelle et de la complexité des réseaux électriques modernes nécessite des modèles de simulation qui offrent non seulement une efficacité de calcul, mais aussi une grande fidélité dans la capture des transitoires et des comportements dépendants de la fréquence. Dans ce contexte, les modèles équivalents de réseau dépendants de la fréquence se présentent comme une stratégie de modélisation prometteuse. Les modèles de réseaux équivalents à dépendance fréquentielle (FDNE) permettent une représentation plus précise de la dynamique des réseaux externes en tenant compte des caractéristiques d'impédance dépendantes de la fréquence, souvent négligées par les modèles classiques. Cependant, la simulation temps réel des FDNE pose des défis computationnels importants, en raison de leur complexité mathématique inhérente et de la nécessité de capturer avec précision les transitoires électromagnétiques rapides.

Ces conditions ont motivé la présente recherche, dont l'objectif est d'établir une méthodologie de conception structurée pour la simulation en temps réel sur FPGA de FDNE. Afin de faciliter l'accessibilité et de réduire la complexité de développement, la synthèse de haut niveau est utilisée pour implémenter les FDNEs sur des plateformes FPGA, permettant ainsi aux ingénieurs d'application de s'abstraire des détails matériels de bas niveau. Toutefois, atteindre des pas de temps inférieurs à la microseconde avec des modèles avancés tels que le FDNE — caractérisés par de larges opérations matricielles et des formulations en espace d'état —

demeure un défi majeur. Cette thèse propose une méthodologie qui répond simultanément à la complexité computationnelle propre à la modélisation de FDNE et aux limitations de performance souvent rencontrées dans les conceptions matérielles basées sur la synthèse de haut niveau. Une attention particulière est portée au rôle de la représentation numérique, ce qui conduit au développement de bibliothèques arithmétiques inédites et de formats de données personnalisés, spécifiquement optimisés pour une utilisation avec des outils de synthèse de haut niveau. L'approche proposée permet d'améliorer la latence, d'accroître l'extensibilité et d'optimiser l'utilisation des ressources. En comblant le fossé entre modélisation de haute fidélité et exécution efficace en temps réel, la méthodologie développée rend possible la simulation sur FPGA de systèmes électriques complexes intégrant de FDNE, tout en respectant les contraintes temporelles strictes imposées par les environnements HIL.

1.3 Objectifs de recherche

L'objectif principal de cette recherche est de proposer une méthodologie de conception structurée pour les simulateurs en temps réel basés sur FPGA, avec un accent particulier sur la simulation des équivalents de réseau dépendants de la fréquence. La méthodologie proposée vise à combler le fossé entre la conception matérielle hautes performances et l'accessibilité nécessaire pour les spécialistes d'application, notamment dans le domaine des systèmes électriques et des environnements HIL. En introduisant des architectures matérielles personnalisables et des bibliothèques dédiées basées sur la synthèse de haut niveau, ce travail permet la mise en œuvre de solveurs efficaces et à faible latence spécifiquement adaptés à la modélisation des FDNE. La méthodologie prend en charge une résolution numérique flexible, une capacité d'évolution adaptée à différents niveaux de complexité des systèmes, ainsi qu'une optimisation de l'utilisation des ressources — facilitant ainsi le développement rapide et le déploiement de plateformes de simulation en temps réel, à la fois précises et efficaces sur FPGA. Afin d'atteindre cet objectif, nous avons :

1. Étudié la complexité des modèles FDNE et leurs formulations mathématiques dans le but de permettre une simulation en temps réel ;
2. Proposé des bibliothèques HLS permettant de prendre en charge des formats numériques personnalisables adaptés à l'accélération matérielle ;
3. Proposé une méthode efficace pour simuler précisément les modèles FDNE ; mis en œuvre et optimisé l'implémentation matérielle du simulateur en temps réel des modèles FDNE.

1.4 Plan de thèse

Ce chapitre a présenté une introduction à la simulation en temps réel, a discuté des défis liés à son développement, et a exposé les principaux objectifs de cette recherche. Les chapitres suivants présentent la structure du reste de ce travail.

— Chapitre 2 : Ce chapitre introduit les concepts fondamentaux liés à la simulation en temps réel. Il présente les formulations mathématiques les plus couramment utilisées ainsi qu'un aperçu de la modélisation des Solveurs d'analyse nodale modifiée (Modified Nodal Analysis, MNA). Une distinction est établie entre les approches de simulation hors ligne et en ligne, et un contexte historique de l'évolution des technologies de simulation en temps réel est également exposé ;

— Chapitre 3 : Ce chapitre passe en revue les travaux les plus étroitement liés à cette recherche. Il présente un aperçu de la modélisation de FDNE et met en évidence les principaux défis associés à leur simulation en temps réel. Le chapitre explore diverses techniques d'implémentation de la simulation en temps réel des FDNE sur des plateformes FPGA, en analysant différentes stratégies architecturales ainsi que leurs limites. Enfin, il présente l'état de l'art en matière de simulation en temps réel ;

— Chapitre 4 : Ce chapitre décrit le processus suivi pour générer et structurer les principaux chapitres qui composent cette thèse ;

— Chapitre 5 : Ce chapitre présente une méthode de simulation en temps réel sur FPGA pour les faisceaux de câbles d'avion, basée sur les modèles de FDNE. L'approche exploite les techniques HIL afin d'atteindre des pas de simulation submicrosecondes, avec une évaluation de différents formats numériques pour garantir précision et efficacité. Les résultats confirment la pertinence de cette méthode pour la simulation à haute fidélité des systèmes électriques aéronautiques. Il s'agit du premier article issu de ce travail [3]. Ce travail a été publié dans les actes de la 12^{e} Conférence Internationale sur la Recherche et les Applications en Énergie Renouvelable (ICRERA) en 2023. À la date de rédaction du présent document, l'article associé a été cité trois fois selon Google Scholar ;

— Chapitre 6 : Ce chapitre présente une bibliothèque de synthèse de haut niveau, nommée CuFP, qui permet l'arithmétique à virgule flottante personnalisable sur FPGA. CuFP offre la possibilité aux utilisateurs de définir la largeur des bits de l'exposant et de la mantisse. La bibliothèque montre des performances améliorées — notamment une réduction de la latence et de l'utilisation des ressources — pour des opérations clés telles que la sommation vectorielle, le produit scalaire et la multiplication matrice-vecteur. Grâce à son architecture basée sur des modèles (templates), CuFP est particulièrement

adaptée aux applications nécessitant des calculs à virgule flottante efficaces et flexibles. Il s'agit du deuxieme article issu de ce travail [4]. Ce travail a été publié dans la revue Electronics, numéro spécial sur les systèmes embarqués reconfigurables basés sur FPGA en 2024. À la date de rédaction du présent document, l'article associé a été cité trois fois selon Google Scholar ;

— Chapitre 7 : Ce chapitre présente un cadre de simulation en temps réel basé sur FPGA pour les convertisseurs connectés au réseau, en utilisant un STATCOM comme cas d'étude et en intégrant un modèle de FDNE pour la modélisation du réseau électrique. La recherche propose et analyse différentes approches d'intégration, tout en appliquant la bibliothèque CuFP afin d'améliorer les performances et d'optimiser l'utilisation des ressources matérielles. Elle vise aussi à fournir un outil pour le test et la validation de stratégies de commande avancées dans les réseaux électriques modernes, en particulier dans le contexte d'une intégration croissante des énergies renouvelables. Ce travail correspond à la troisième publication de cette thèse, publiée dans 16^{e} édition de la Conférence internationale sur les transitoires des systèmes électriques en 2025 [5] ;

— Chapitre 8 : Ce chapitre présente CuFPSAF, une version améliorée de la bibliothèque CuFP intégrant la technique d'auto-alignement (SAT) pour optimiser les opérations en virgule flottante sur FPGA. Cette approche permet d'améliorer à la fois la précision et l'utilisation des ressources. Les évaluations de performance menées sur la sommation vectorielle, le produit scalaire et la multiplication matrice-vecteur montrent des réductions significatives de la latence et de la consommation matérielle, en particulier pour les réseaux de grande taille. Ce travail correspond à la quatrième publication de cette thèse, publiée dans Computers en 2025 [6] ;

— Chapitre 9 : Ce chapitre examine des techniques visant à améliorer l'efficacité des additionneurs sur FPGA, éléments essentiels dans les applications de calcul à haute vitesse. Un additionneur High-Radix Carry-Save (HRCS) amélioré est présenté, intégrant une nouvelle architecture de chaîne de retenue spécialement conçue pour réduire la latence de calcul — un facteur déterminant pour les performances en simulation temps réel. Ce travail correspond à la cinquième publication de cette thèse, présentée à la 23^{e} conférence internationale IEEE NEWCAS en 2025 [7] ;

— Chapitre 10 : Ce chapitre présente une discussion générale des approches méthodologiques et des résultats obtenus dans le cadre de ce travail de recherche ;

— Chapitre 11 : Conclusion et recommandations : Ce dernier chapitre résume les résultats clés et met en lumière les principaux enseignements tirés de cette thèse. Il se termine par des suggestions de pistes de recherche future afin de poursuivre et approfondir les travaux engagés.

1.5 Contributions de la thèse

Cette recherche a apporté plusieurs contributions scientifiques et techniques, réparties entre le développement méthodologique, la conception de bibliothèques matérielles spécialisées et l'évaluation expérimentale sur FPGA. Les contributions principaux peuvent être résumés comme suit :

- Proposition d'une méthodologie de conception de simulateurs temps réel sur FPGA : un cadre structuré et de haut niveau a été introduit, spécifiquement dédié à la simulation en temps réel des équivalents de réseau dépendants de la fréquence. Cette méthodologie vise à combler le fossé entre la complexité computationnelle de la modélisation des réseaux électriques et les contraintes strictes du temps réel, notamment dans les environnements HIL ;
- Étude approfondie de l'impact des formats numériques : une évaluation comparative a été menée entre différents types de données — virgule flottante (simple et double précision), virgule fixe et formats flottants personnalisés — afin d'identifier les compromis entre précision, latence et consommation de ressources ;
- Développement de bibliothèques dédiées à la synthèse de haut niveau : deux bibliothèques ont été proposées, CuFP et CuFPSAF, permettant la configuration flexible des formats flottants et l'alignement efficace des exposants. Ces outils ont montré des gains notables en performance et en consommation de ressources par rapport aux solutions standards du fournisseur ;
- Amélioration d'architectures matérielles fondamentales : une nouvelle architecture d'additionneur, nommée IESC-HRCS, a été introduite. Elle réduit significativement la latence dans les opérations arithmétiques intensives tout en maintenant une utilisation comparable des ressources ;
- Validation expérimentale sur des cas d'étude représentatifs : la méthodologie et les bibliothèques proposées ont été implémentées et validées sur FPGA à travers des études de cas en aéronautique (réseaux électriques embarqués) et en électronique de puissance (convertisseurs connectés au réseau). Les résultats ont démontré une exécution inférieure à la microseconde, une grande précision numérique et une scalabilité vers des modèles de plus grande taille.

Cette thèse a donné lieu à la publication de plusieurs articles de conférence et de journaux avec comité de lecture, dont voici la liste :

1. Article 1 : FPGA-based FDNE Models for the Accurate Real-time Simulation of Power Systems in Aircrafts [8]

2. Article 2 : CuFP : An HLS Library for Customized Floating-Point Operators [4]
3. Article 3 : FPGA-based Simulation of Grid-tied Converters using Frequency-dependent Network Equivalent [5]
4. Article 4 : Enhancing CuFP Library with Self-Alignment Technique [6]
5. Article 5 : Improved High-Radix Carry-Save Adders for Low-Latency Vector Reduction on FPGA [7]

CHAPITRE 2 CONTEXTE

2.1 Introduction

L'utilisation de la simulation en temps réel est devenue une approche essentielle dans le développement et la validation des systèmes électriques modernes. Elle joue un rôle déterminant dans divers secteurs de haute technologie, tels que les réseaux intelligents, l'intégration des énergies renouvelables, la mobilité électrique et les technologies aérospatiales. En reproduisant le comportement des systèmes en temps réel, les ingénieurs peuvent analyser les performances, identifier les dysfonctionnements potentiels à un stade précoce et affiner les stratégies de commande bien avant la mise en œuvre physique. Ce processus permet ainsi de réduire les coûts de développement tout en renforçant la fiabilité des systèmes.

L'évolution de cette technologie a débuté avec les simulateurs analogiques, tels que le Transient Network Analyzer (TNA), dans lesquels les systèmes électriques étaient modélisés à l'aide de composants physiques à échelle réduite. Ces premières plateformes ont permis de réaliser des essais pratiques sur le matériel de commande et les dispositifs de protection, notamment via des configurations HIL. Par la suite, les progrès en informatique ont permis l'émergence de simulateurs basés sur des architectures CPU, offrant une flexibilité accrue et une modélisation plus rapide. L'introduction des FPGA a ensuite profondément transformé le domaine : d'abord en tant qu'outils d'interface performants, puis en tant qu'unités de calcul parallèles capables de traiter des simulations avec des pas de temps inférieurs à la microseconde. Ces avancées ont considérablement élargi le champ d'application et la précision des simulations en temps réel, tant dans l'industrie que dans le milieu académique.

Dans ce chapitre, nous abordons de manière concise les principaux concepts, approches, modèles et outils associés à la conception et à l'utilisation de simulateurs temps réel pour réseaux électriques, en particulier ceux basés sur FPGA.

2.2 Équations des circuits électriques

La simulation moderne des réseaux électriques répond à divers besoins techniques, allant de la coordination des protections et l'analyse harmonique à la planification de la production et l'optimisation des réseaux de distribution. Ces outils numériques permettent d'étudier différents scénarios opérationnels, incluant les conditions de défaut, le comportement dynamique des machines et la performance des systèmes de contrôle. Ce travail se concentre spécifiquement sur la simulation des transitoires électromagnétiques (Electromagnetic Transient,

EMT), qui analyse la réponse temporelle du réseau aux perturbations rapides.

Le processus de simulation commence par la représentation mathématique du système physique. Les réseaux électriques sont généralement modélisés à l'aide de formulations discrètes combinant des équations différentielles et algébriques (Differential and Algebraic Equations, DAEs). Cette représentation hybride capture à la fois les aspects dynamiques et statiques du comportement du système. Un environnement de simulation EMT efficace doit accomplir trois tâches essentielles : (1) établir le système complet d'équations, (2) appliquer des méthodes de discrétisation appropriées et (3) résoudre numériquement le système d'équations résultant.

2.3 Analyse nodale classique

L'analyse nodale est une méthode systématique de formulation des équations de circuits électriques, fondée sur la loi des nœuds de Kirchhoff. Selon cette loi, la somme algébrique des courants entrant dans un nœud est égale à celle des courants qui en sortent. Cette approche est couramment employée dans l'analyse des circuits linéaires et non linéaires en raison de sa structure formelle et de son efficacité computationnelle.

Le principe de l'analyse nodale consiste à exprimer les courants de branche en fonction des tensions nodales, ce qui conduit à une représentation matricielle du système :

$$\mathbf{Yv} = \mathbf{i}, \tag{2.1}$$

Dans cette équation, $\mathbf{v}$ représente le vecteur des tensions nodales inconnues, mesurées par rapport au nœud de référence, également appelé masse. Le vecteur $\mathbf{i}$ correspond aux courants injectés dans chaque nœud du circuit, tandis que $\mathbf{Y}$ est la matrice d'admittance, qui contient les conductances des différents éléments du circuit.

2.4 Méthode d'analyse nodale modifiée et augmentée

L'analyse nodale modifiée (Modified Nodal Analysis, MNA) s'est imposée comme une technique fondamentale en simulation de circuits, reconnue pour son approche systématique de la modélisation des réseaux électriques. Bien qu'elle ait été initialement développée pour l'analyse de circuits électroniques, cette méthode a été adaptée avec succès aux systèmes électriques de puissance, où elle est souvent désignée sous le nom d'analyse nodale modifiée et augmentée (Modified and Augmented Nodal Analysis, MANA) [9]. Cette adaptation fournit un cadre unifié et complet permettant de représenter une large variété de composants des

systèmes de puissance ainsi que leurs interconnexions complexes.

La méthode nodale classique présente des limitations notables lorsqu'elle est confrontée à des sources de tension indépendantes, en particulier lorsque les deux bornes de la source ne sont pas reliées au nœud de référence. La méthode MANA surmonte efficacement ces difficultés grâce à une formulation augmentée, qui permet de compléter systématiquement les équations du système. La représentation complète prend la forme suivante :

$$\mathbf{A}\mathbf{x}(t) = \mathbf{b}(t) \tag{2.2}$$

où $\mathbf{A}$ désigne la matrice du système MANA, $\mathbf{x}(t)$ contient les tensions nodales inconnues ainsi que certaines courants de branche sélectionnés, et $\mathbf{b}(t)$ regroupe les sources connues ainsi que les termes historiques. La formulation détaillée de la matrice s'écrit comme suit :

$$\begin{bmatrix} \mathbf{Y} & \mathbf{V}_c & \mathbf{D}_c & \mathbf{S}_c \\ \mathbf{V}_r & 0 & 0 & 0 \\ \mathbf{D}_r & 0 & 0 & 0 \\ \mathbf{S}_r & 0 & 0 & \mathbf{S}_d \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{i}_V \\ \mathbf{i}_D \\ \mathbf{i}_S \end{bmatrix} = \begin{bmatrix} \mathbf{i} \\ \mathbf{v}_b \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{2.3}$$

Dans cette formulation, la matrice $\mathbf{Y}$, le vecteur $\mathbf{v}$ et le vecteur $\mathbf{i}$ représentent respectivement la matrice d'admittance nodale, les tensions nodales inconnues et les courants injectés de l'analyse nodale classique. Les indices $\mathbf{r}$, $\mathbf{c}$ et $\mathbf{d}$ désignent respectivement les composantes ligne, colonne et diagonale des matrices.

La sous-matrice $\mathbf{V}_r$ décrit les connexions des sources de tension, tandis que $\mathbf{D}_r$ intègre les équations des branches dépendantes, telles que celles des transformateurs idéaux. Les sous-matrices $\mathbf{S}$ modélisent les interrupteurs idéaux, avec $\mathbf{S}_d$ représentant spécifiquement l'état de commutation. Les matrices colonnes correspondent aux transposées des matrices lignes associées, et les autres sous-matrices sont fréquemment nulles dans les applications pratiques.

2.5 Méthode de représentation de l'espace d'état

La représentation dans l'espace d'état constitue un cadre fondamental pour la modélisation des systèmes linéaires invariants dans le temps (Linear Time-Invariant, LTI), en particulier ceux décrits par des équations différentielles du premier ordre. Pour un tel système, la forme usuelle s'écrit :

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{cases} \tag{2.4}$$

Dans cette formulation, $\mathbf{x}$ désigne le vecteur des variables d'état internes, $\mathbf{u}$ correspond aux entrées du système, et $\mathbf{y}$ représente les sorties. La matrice $\mathbf{A}$ décrit l'évolution dynamique du système, $\mathbf{B}$ traduit l'influence des entrées sur les états, $\mathbf{C}$ relie les états aux sorties, et $\mathbf{D}$ modélise une interaction directe entre les entrées et les sorties.

Lorsque le pas de temps utilisé pour la simulation est bien inférieur aux constantes de temps caractéristiques du système, une méthode d'intégration implicite de faible ordre comme la méthode d'Euler implicite (Backward Euler) s'avère appropriée pour discrétiser les équations d'état. On obtient alors :

$$\begin{cases} \mathbf{x}(t) = \mathbf{A}_d\mathbf{x}(t - \Delta t) + \mathbf{B}_d\mathbf{u}(t) \\ \mathbf{y}(t) = \mathbf{C}_d\mathbf{x}(t - \Delta t) + \mathbf{D}_d\mathbf{u}(t) \end{cases} \tag{2.5}$$

avec les matrices discrètes définies par :

$$\begin{cases} \mathbf{A}_d = (\mathbf{I} - \Delta t\,\mathbf{A})^{-1} \\ \mathbf{B}_d = (\mathbf{I} - \Delta t\,\mathbf{A})^{-1}\Delta t\,\mathbf{B} \\ \mathbf{C}_d = \mathbf{C}(\mathbf{I} - \Delta t\,\mathbf{A})^{-1} \\ \mathbf{D}_d = \mathbf{D} + \mathbf{C}(\mathbf{I} - \Delta t\,\mathbf{A})^{-1}\Delta t\,\mathbf{B} \end{cases} \tag{2.6}$$

Ici, $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$ et $\mathbf{D}_d$ sont les matrices discrètes issues de la formulation continue. $\mathbf{I}$ est la matrice identité.

Pour une implémentation optimisée sur des architectures parallèles telles que les FPGAs, cette équation peut être réécrite sous forme d'un produit matrice-vecteur :

$$\begin{bmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}(t - \Delta t) \\ \mathbf{u}(t) \end{bmatrix} \tag{2.7}$$

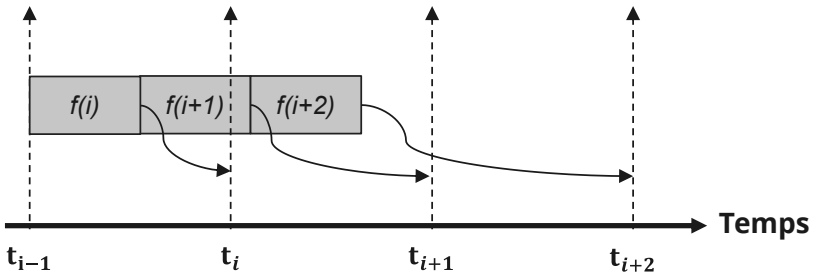
Cette formulation facilite l'exploitation du parallélisme inhérent aux FPGAs, permettant des simulations temps réel avec des pas de temps très réduits.

2.6 Simulation en temps réel

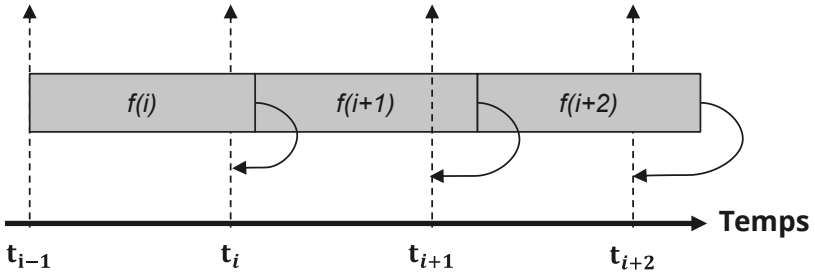
La simulation en temps réel joue un rôle essentiel dans l'analyse, le test et la validation des systèmes dynamiques complexes en reproduisant leur comportement avec une grande fidélité. Elle est largement utilisée dans diverses applications d'ingénierie, notamment les systèmes électriques, les systèmes de contrôle et les tests de matériel embarqué. En maintenant une synchronisation avec la dynamique réelle du système, la simulation en temps réel permet une évaluation précise des performances dans des conditions réelles. La simulation numérique en temps réel fonctionne sur une base de pas de temps discret, où le simulateur résout successivement les équations du modèle. Dans ce type de simulation, le temps nécessaire pour calculer toutes les équations et fonctions représentant un système pendant un pas de temps donné peut être soit plus court, soit plus long que la durée réelle du pas de simulation. La Figure 2.1 illustre ces deux cas de figure.

Dans une simulation en temps différé, les calculs sont effectués indépendamment des contraintes de temps réel. Comme illustré dans la Figure 2.1a, cela peut entraîner des temps d'exécution soit plus rapides (1), soit plus lents (2) que le temps réel. L'objectif principal de la simulation en temps différé est d'obtenir des résultats très précis dans les plus brefs délais. La vitesse de simulation dépend de plusieurs facteurs, notamment la complexité du modèle mathématique du système. En revanche, la simulation en temps réel garantit que chaque étape de calcul est complétée dans la même durée que le pas de temps réel correspondant, comme le montre la Figure 2.1b. Ici, l'exécution de chaque pas de temps est synchronisée avec le temps réel. Pour être valide, une simulation en temps réel doit produire des variables internes et des sorties précises dans la même durée que le système physique. Cette synchronisation permet au simulateur d'effectuer toutes les opérations nécessaires tout en maintenant la pertinence en temps réel [10].

Les simulateurs en temps réel jouent un rôle clé à différentes étapes du processus de conception, permettant d'évaluer les performances du système dans diverses conditions de fonctionnement [11]. La courbe en V, représentée à la Figure 2.2, illustre l'utilisation de la simulation en temps réel tout au long des différentes phases de développement. Lors de la phase de conception, la vitesse de calcul élevée des simulateurs en temps réel accélère le processus de simulation, réduisant ainsi le temps de développement. Au cours de la phase de prototypage, ces simulateurs permettent de tester la fonctionnalité du prototype avant de passer aux étapes suivantes, ce qui permet d'identifier les éventuels problèmes à un stade précoce et de minimiser les coûts et le temps de développement. De plus, les simulateurs en temps réel sont utilisés dans les phases d'implémentation, de vérification et de test afin d'analyser le comportement du système dans différentes conditions d'exploitation.

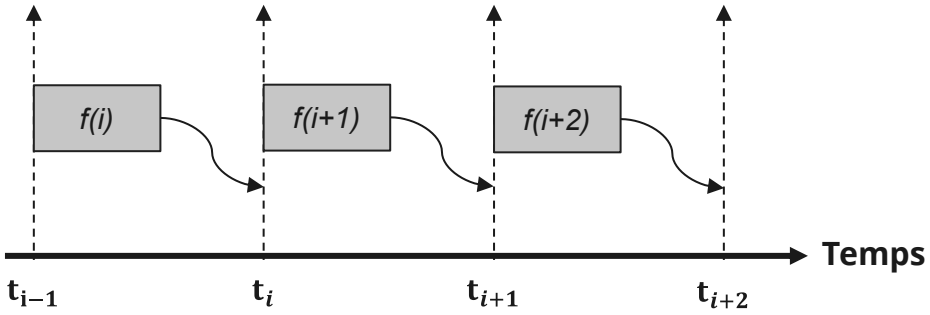


(1)



(2)

(a) Simulation en temps différé (pas de synchronisation) : (1) Plus rapide que le temps réel, (2) Plus lent que le temps réel



(b) Simulation en temps réel (synchronisée)

FIGURE 2.1 (a) Simulation en temps différé (pas de synchronisation), (b) Simulation en temps réel (synchronisée).

2.7 Évolution de la simulation en temps réel

La simulation en temps réel a considérablement évolué au cours des dernières décennies. À l'origine, la simulation physique reposait sur l'utilisation de circuits analogiques comme principal moyen de test et de validation des systèmes complexes. Cependant, ces simulations

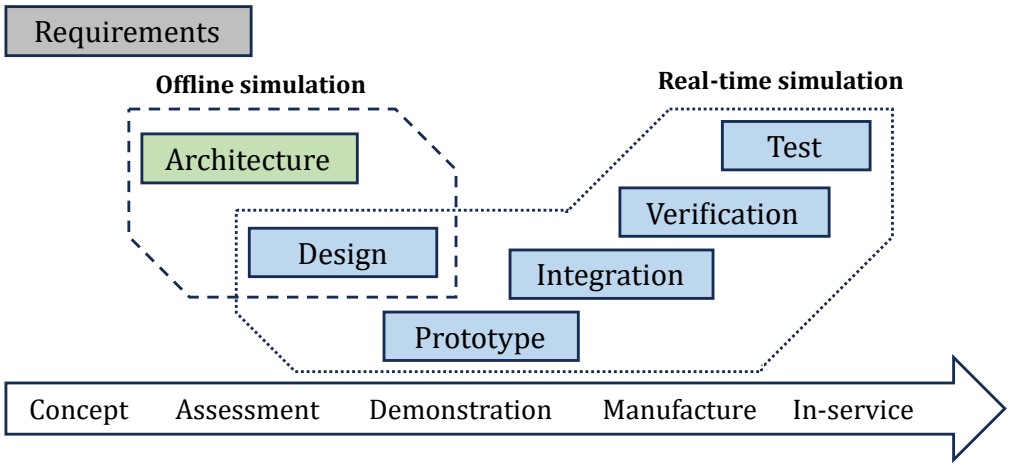


FIGURE 2.2 Utilisation des simulateurs en temps réel aux différentes étapes du processus (inspiré de [12]).

analogiques présentaient plusieurs limitations, notamment des coûts élevés et des délais de développement importants [12].

L'émergence des simulateurs numériques en temps réel dans les années 1980 a marqué une transition majeure par rapport aux approches physiques [13]. Malgré leurs avantages, les premiers simulateurs numériques ont rencontré des défis en matière d'évolutivité, leurs capacités de calcul étant limitées par la puissance de traitement disponible. Afin de surmonter ces contraintes, les ingénieurs ont adopté une approche de répartition des tâches de calcul sur plusieurs processeurs numériques de signal (DSP). Avec le développement continu de processeurs centraux (CPU) plus puissants, la simulation en temps réel a progressivement migré vers des architectures basées sur les CPU. Cette évolution a permis d'améliorer la flexibilité, de réduire les coûts et d'optimiser de manière significative les performances globales des systèmes de simulation en temps réel.

Vers la fin des années 1990, les FPGA ont été intégrés aux architectures basées sur des processeurs, améliorant considérablement les capacités des simulateurs en temps réel. Initialement, les FPGA servaient principalement de composants d'interface efficaces pour les systèmes de simulation en temps réel. Cependant, leur rôle a évolué lorsqu'ils ont commencé à être utilisés comme unités de calcul parallèle à haute performance, permettant des simulations plus avancées et plus intensives en calcul dans diverses applications, y compris la simulation en temps réel [14]. Récemment, les FPGA se sont imposés comme une solution prometteuse pour réduire le temps d'aller-retour dans les simulations HIL [15, 16]. Ils offrent un traitement des données à haute vitesse et à faible latence, permettant un retour d'information plus rapide et une émulation précise des conditions réelles [17, 18]. Les FPGA sont reconnus pour leur

efficacité dans les opérations mathématiques et leur structure flexible en termes de taille de mot, ce qui permet des calculs précis sans coût supplémentaire en ressources. Ainsi, en exploitant l'architecture parallèle des FPGA, les équations du système peuvent être implémentées avec un temps d'exécution réduit [19–21]. De plus, les FPGA sont idéaux pour l'interfaçage avec des plateformes de contrôle physique dans le cadre de simulations en temps réel [12, 22]. L'article [23] présente le développement d'un système de simulation en temps réel basé sur FPGA. L'essor des FPGA dans la simulation en temps réel a souligné la nécessité de maîtriser les HDL (Hardware Description Languages). Toutefois, leur apprentissage représente un défi majeur et exige un effort de conception substantiel. Pour contourner ces difficultés, l'approche de HLS s'est imposée comme une solution alternative [24, 25]. La section suivante présente une explication détaillée de cette approche.

2.8 Conclusion

Ce chapitre a présenté les concepts fondamentaux nécessaires à la compréhension de la simulation en temps réel des systèmes électriques, en mettant particulièrement l'accent sur les méthodes adaptées à l'implémentation sur FPGA. Nous avons commencé par la formulation des équations des circuits électriques, puis nous avons abordé l'analyse nodale classique et modifiée. Ensuite, nous avons discuté de la modélisation de l'espace d'état et de sa discrétisation numérique, permettant une simulation en domaine temporel efficace et stable. En conclusion, nous avons présenté un résumé des principes de la simulation en temps réel basée sur FPGA, en soulignant ses avantages en termes de parallélisme, de faible latence et de contrôle précis du timing. Ces concepts forment la base théorique sur laquelle repose le cadre de simulation proposé.

Ce travail se concentre sur la simulation en temps réel basée sur FPGA des convertisseurs de puissance raccordés au réseau, avec un accent particulier sur l'intégration des modèles FDNE. Ainsi, le chapitre suivant présente les méthodologies récentes utilisées pour implémenter la simulation en temps réel basée sur FPGA des convertisseurs raccordés au réseau, en mettant un accent particulier sur la modélisation FDNE et son intégration dans le flux de travail de simulation.

CHAPITRE 3 REVUE DE LITTÉRATURE

3.1 Introduction

Au fil des années, la méthodologie de simulation des réseaux électriques en temps réel a connu une transformation significative. À l'origine, des maquettes physiques analogiques et à échelle réduite étaient utilisées pour reproduire le comportement des systèmes. Toutefois, ces approches manquaient de flexibilité et de capacité d'adaptation. L'avènement du calcul numérique a permis l'émergence des simulations logicielles basées sur CPU, devenues la norme. Néanmoins, le parallélisme limité et les délais de communication importants propres aux systèmes basés sur CPU ont représenté des obstacles pour atteindre des simulations à faible latence et haute résolution temporelle. L'utilisation des FPGA dans les simulateurs temps réel a permis de surmonter bon nombre de ces limitations, en offrant une exécution déterministe et un traitement parallèle efficace. Les travaux récents ont porté sur des architectures hybrides combinant CPU et FPGA, afin de concilier flexibilité et performance. Plus récemment, des simulateurs entièrement basés sur FPGA ont vu le jour, particulièrement adaptés aux applications nécessitant une précision temporelle fine, telles que la modélisation des FDNE. Ce dernier axe représente le cœur de la présente recherche.

Le développement d'un simulateur en temps réel débute généralement par la formulation de modèles mathématiques et d'algorithmes numériques capables de représenter avec précision le comportement dynamique du réseau électrique cible. À ce stade, des plateformes de simulation hors ligne telles que EMTP-RV ou MATLAB/Simulink sont couramment utilisées pour valider les performances attendues du système et affiner les paramètres du modèle. Une fois la vérification terminée, la portion du réseau destinée à être simulée sur FPGA est isolée, en tenant compte des goulots d'étranglement computationnels et des contraintes du temps réel. Les principaux compromis de conception consistent à respecter les échéances temporelles, à minimiser les erreurs d'approximation, et à gérer efficacement les ressources disponibles sur le FPGA (par exemple, les éléments logiques, les blocs DSP et la bande passante mémoire). Cependant, la mise en œuvre des FDNE sur FPGA introduit plusieurs complexités supplémentaires :

- Formulation de modèles et d'algorithmes numériques pour les FDNE
  - Modélisation dépendante de la fréquence : les FDNE nécessitent une représentation précise de l'impédance/admittance dynamique sur une large gamme de fréquences, utilisant souvent des approximations de fonctions rationnelles telles que le vector

fitting.

  - Implémentation du domaine temporel : le modèle dépendant de la fréquence doit être transformé en un équivalent du domaine temporel comme une représentation de l'espace d'état pour l'exécution FPGA.

- Sélection de la méthode de discrétisation
  - Contraintes en temps réel : la méthode choisie, comme l'intégration trapézoïdale ou l'Euler rétrograde, doit équilibrer la précision avec l'exécution par étapes à temps fixe.
  - Stabilité numérique : Les systèmes électriques impliquent souvent des équations rigides, notamment lors de la modélisation de composants tels que des transformateurs ou des câbles. Les méthodes Euler rétrograde et trapézoïdale offrent une stabilité A et une stabilité L, ce qui les rend préférables dans de tels scénarios.
- Défis de mise en œuvre du FPGA
  - Parallélisme et pipelining : les FPGAs excellent dans l'exploitation du parallélisme. Les modèles FDNE, en particulier sous forme d'équations d'état, bénéficient des multiplications matrice-vecteur en pipeline et de l'évaluation parallèle de plusieurs pôles. Cependant, les dépendances liées aux boucles, comme les termes historiques récursifs, doivent être planifiées avec soin pour éviter les goulots d'étranglement en termes de latence.
  - Résolution numérique : les représentations numériques, telles que l'arithmétique en virgule flottante ou en virgule fixe, influencent fortement la précision des calculs, la latence et l'utilisation des ressources matérielles. La virgule flottante offre une large plage dynamique et une grande précision, mais elle est plus coûteuse en ressources. À l'inverse, la virgule fixe est mieux adaptée aux FPGAs, mais nécessite une optimisation rigoureuse de la longueur des mots pour éviter les dépassements ou les pertes de précision. Le choix du format et de la précision appropriés est essentiel pour atteindre un compromis optimal entre performance, exactitude et contraintes matérielles dans l'implémentation temps réel des FDNE.
  - Utilisation des ressources : les opérations matricielles, les mises à jour des termes historiques et la gestion des entrées/sorties consomment des blocs DSP et des LUTs. Des techniques de partage de ressources et de multiplexage temporel peuvent être mises en œuvre si plusieurs blocs FDNE doivent coexister sur un même FPGA.
- Vérification et validation
  - Validation hors ligne : avant son déploiement, le modèle FDNE sur FPGA doit être validé par comparaison avec des outils logiciels de haute fidélité comme EMTP-RV

ou MATLAB/Simulink afin de garantir son exactitude.

- Robustesse en régime transitoire : le système doit rester stable en présence de variations brusques telles que des défauts, des commutations de lignes ou le fonctionnement de convertisseurs. Une attention particulière doit être portée au maintien de l'encadrement des états internes dans des conditions extrêmes.

— Intégration avec les plateformes de simulation temps réel

- Échange de données et interfaces : les modules FDNE doivent s'interfacer de manière transparente avec d'autres blocs FPGA ou CPU, comme les modèles STATCOM ou les systèmes de commande. Des interfaces telles que AXI-Stream, DMA ou des bus personnalisés doivent être mises en œuvre pour assurer un transfert de données efficace avec une latence minimale.
- Scalabilité : l'architecture doit être conçue pour accueillir plusieurs instances FDNE, afin de supporter des systèmes à grande échelle ou des configurations multi-ports.

Ce chapitre passe en revue les avancées actuelles des simulateurs en temps réel basés sur FPGA pour FDNE, en mettant en lumière les limites des solutions existantes et en situant notre recherche dans ce contexte. Il présente les stratégies de conception utilisées pour implémenter les simulations en temps réel de FDNE sur FPGA, notamment l'application de la synthèse de haut niveau, l'évaluation des différentes méthodes de résolution numérique et des approches FDNE. Le chapitre se termine par une comparaison des travaux les plus pertinents dans le domaine, en soulignant leur relation avec les objectifs de cette recherche.

3.2 Réseaux équivalents à dépendance fréquentielle (Frequency Dependent Network Equivalent, FDNE)

Les systèmes électriques modernes sont de grande envergure et fortement interconnectés, impliquant souvent des interactions complexes entre divers composants. Pour simuler efficacement de tels systèmes, en particulier dans les études de type EMT, il est souvent trop complexe de modéliser l'ensemble du réseau dans le moindre détail. Les modèles FDNE offrent une représentation compacte mais précise des portions externes du réseau en capturant leur comportement en domaine fréquentiel sur une large plage de fréquences. Cela permet de réduire la complexité du modèle tout en conservant une bonne fidélité dans les simulations transitoires. Les FDNE sont particulièrement utiles dans les simulations en temps réel, où l'efficacité computationnelle est essentielle sans compromettre la précision de la réponse dynamique du système. Quelques avantages des FDNE sont listés ci-dessous :

— Comportement dépendant de la fréquence : les réseaux électriques présentent des ré-

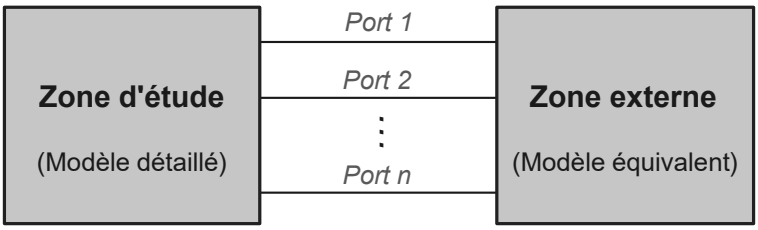


FIGURE 3.1 Partitionnement d'un système électrique en une zone d'étude et une zone externe

ponses différentes selon la fréquence. Le FDNE prend en compte les variations d'impédance ou d'admittance sur une plage de fréquences, ce qui permet une représentation plus précise du réseau.

— Effets d'impédance complexe : les réseaux peuvent avoir un comportement d'impédance complexe, incluant des composantes inductives, capacitives et résistives. Le FDNE tient compte de ces effets pour modéliser le réseau de manière plus réaliste.

— Analyse harmonique : dans les systèmes électriques, les harmoniques sont des multiples de la fréquence fondamentale qui peuvent provoquer des distorsions et altérer les performances globales. Les FDNE permettent de capturer les effets des harmoniques, ce qui est important dans les systèmes comportant des dispositifs de commutation, des défauts ou de l'électronique de puissance.

Cependant, il est crucial de reconnaître les limitations et les inconvénients des FDNE. Le principal défi réside dans leur complexité et leurs exigences computationnelles élevées. En effet, les modèles FDNE nécessitent des caractéristiques détaillées du système en fonction de la fréquence, sur une large gamme de fréquences. Ces informations doivent être obtenues par des campagnes de mesures approfondies ou des analyses détaillées, ce qui peut s'avérer long et exigeant en ressources. De plus, la charge computationnelle associée à la résolution des modèles FDNE augmente avec le nombre de points fréquentiels pris en compte, ce qui constitue un défi majeur pour les applications de simulation en temps réel.

Une autre limitation majeure des modèles FDNE réside dans leur incapacité à modéliser avec précision les composants et comportements non linéaires. En effet, des éléments tels que les diodes et les transistors, dont les dynamiques non linéaires jouent un rôle crucial dans les performances du système, ne peuvent pas être correctement représentés par les approximations FDNE. Cela peut entraîner des imprécisions significatives dans les résultats de simulation lorsque les effets non linéaires dominent.

Comme illustré dans la Figure 3.1, le système est divisé en deux zones : (i) une zone d'étude (région d'intérêt principal) et (ii) une zone externe (représentée par le modèle FDNE). L'ex-

clusion des non-linéarités dans la zone externe peut compromettre davantage la précision de la simulation, en particulier si les interactions entre les deux zones impliquent des comportements fortement non linéaires. Le système est divisé en deux zones afin d'équilibrer la précision de la simulation et l'efficacité computationnelle. La zone d'étude contient une représentation détaillée des composants dominant les phénomènes transitoires, garantissant ainsi une simulation précise dans la plage de fréquences ciblée. La zone externe, quant à elle, regroupe les composants restants du système, qui sont modélisés par un FDNE valide sur un large spectre de fréquences. La plage fréquentielle requise pour le modèle FDNE dépend du phénomène spécifique analysé.

3.3 Modélisation FDNE

L'idée fondamentale de la modélisation FDNE est de reproduire la réponse fréquentielle d'un réseau électrique à ses ports de frontière. Un FDNE capture ce comportement selon la relation suivante :

$$\mathbf{i} = \mathbf{Y}(s)\mathbf{v} \tag{3.1}$$

où $\mathbf{i}(s)$ et $\mathbf{v}(s)$ représentent respectivement les vecteurs de courants et de tensions aux ports dans le domaine de Laplace, et $\mathbf{Y}(s)$ est la matrice d'admittance dépendante de la fréquence de taille $p \times p$, où p est le nombre de ports du réseau :

$$\mathbf{Y}(s) = \begin{bmatrix} y_{11}(s) & \cdots & y_{1p}(s) \\ \vdots & \ddots & \vdots \\ y_{p1}(s) & \cdots & y_{pp}(s) \end{bmatrix} \tag{3.2}$$

Afin de permettre une simulation dans le domaine temporel, $\mathbf{Y}(s)$ est approchée par une décomposition partielle :

$$\mathbf{Y}(s) \approx \mathbf{Y}_{fitted}(s) = \mathbf{G}_0 + s\mathbf{E} + \sum_{k=1}^{n} \frac{\mathbf{R}_k}{s - p_k} \tag{3.3}$$

où p_k et $\mathbf{R}_k$ sont respectivement les pôles et les matrices résiduelles du modèle, n est le nombre total de pôles, et $\mathbf{G}_0$ ainsi que $\mathbf{E}$ sont des matrices constantes (généralement, $\mathbf{E}$ est nulle). Cet ajustement est couramment réalisé à l'aide de l'algorithme d'ajustement vectoriel (vector fitting), bien établi dans la recherche académique et les applications industrielles [26].

Une méthode d'intégration numérique, telle que l'Euler implicite ou la méthode trapézoïdale, est ensuite appliquée pour convertir cette représentation en temps continu, fournissant un équivalent Norton donné par :

$$\mathbf{i}(t) = \mathbf{G}_Y \mathbf{v}(t) + \mathbf{i}^h(t) \tag{3.4}$$

où $\mathbf{G}_Y$ est la matrice de conductance associée à $\mathbf{G}_0$, et $\mathbf{i}^h(t)$ est le terme de courant dépendant de l'historique, défini par :

$$\mathbf{i}^h(t) = \sum_{k=1}^{n} \mathbf{x}_k(t) \tag{3.5}$$

Chaque composante $\mathbf{x}_k(t)$ évolue au cours du temps selon la relation récursive :

$$\mathbf{x}_k(t) = \alpha_k \mathbf{x}_k(t - \Delta t) + \beta_k \mathbf{v}(t - \Delta t) \tag{3.6}$$

Par exemple, avec la méthode trapézoïdale, les coefficients sont :

$$\alpha_k = \frac{2 + p_k \Delta t}{2 - p_k \Delta t}, \quad \beta_k = \frac{4 \Delta t \mathbf{R}_k}{(2 - p_k \Delta t)^2} \tag{3.7}$$

et pour la méthode d'Euler implicite :

$$\alpha_k = \frac{1}{1 - p_k \Delta t}, \quad \boldsymbol{\beta}_k = \frac{\Delta t \boldsymbol{R}_k}{(1 - p_k \Delta t)^2} \tag{3.8}$$

Ici, $\mathbf{x}_k(t)$ et $\mathbf{x}_k(t - \Delta t)$ représentent les vecteurs d'états internes actuels et précédents associés à la paire pôle-résidu $k^{\text{ème}}$.

Le modèle FDNE peut également être formulé à l'aide d'équations d'état comme suit :

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \mathbf{i}_F(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}_F(t) \end{bmatrix} \tag{3.9}$$

où $\mathbf{x}(t)$ désigne la variable d'état du FDNE. Les matrices $\mathbf{A}$ et $\mathbf{C}$ contiennent respectivement les pôles et les résidus du modèle, avec des dimensions $(pn) \times (pn)$ et $p \times (pn)$. La matrice $\mathbf{B}$ est une matrice clairsemée contenant des uns et des zéros qui fait le lien entre les entrées de tension et les dérivées d'états, de taille $(pn) \times p$. La matrice $\mathbf{D}$ est identique à celle de l'équation 3.3, de dimension $p \times p$. Les vecteurs d'entrée et de sortie $\mathbf{v}_F(t)$ et $\mathbf{i}_F(t)$ représentent respectivement les tensions et courants aux ports.

Une méthode d'intégration numérique est ensuite appliquée pour obtenir la forme discrète :

$$\begin{bmatrix} \mathbf{x}(t+\Delta t) \\ \mathbf{i}_F(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{3.10}$$

où $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$ et $\mathbf{D}_d$ sont les versions discrètes respectives des matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ et $\mathbf{D}$.

3.4 Méthodes en temps réel basées sur FPGA pour la simulation des FDNE

La simulation en temps réel des FDNEs est devenue de plus en plus cruciale pour l'analyse des réseaux électriques. Trois facteurs principaux expliquent cette importance : la nécessité de valider les modèles, le besoin de modéliser avec précision les dynamiques des convertisseurs électroniques de puissance, et l'exigence d'une simulation à haute fidélité des transitoires électromagnétiques rapides. Pour répondre à ces exigences élevées, diverses méthodes et stratégies d'optimisation ont été développées. Celles-ci visent principalement à améliorer le parallélisme des calculs, à réduire le pas de temps de la simulation et à simplifier l'implémentation des simulateurs en temps réel. Les capacités uniques des FPGAs les rendent particulièrement adaptés aux applications de simulation des FDNEs. En permettant un calcul efficace des équations complexes du réseau tout en maintenant des vitesses d'exécution en temps réel, ces dispositifs surmontent les limites des simulateurs traditionnels. De plus, leur aptitude à préserver la précision numérique lors de pas de temps inférieurs à la microseconde, ainsi que leur fiabilité dans des configurations de type HIL, en font la plateforme privilégiée pour les simulations exigeantes des systèmes électriques.

3.5 Synthèse de haut niveau (High-Level Synthesis, HLS)

La conception au niveau RTL est fondamentale pour décrire les circuits numériques à l'aide de langages de description matérielle (HDL) tels que Verilog et VHDL. Elle influence directement l'efficacité, les performances et la fiabilité du système. Cependant, à mesure que les conceptions numériques deviennent de plus en plus complexes, le développement RTL pose des défis majeurs, nécessitant une connaissance approfondie de l'architecture matérielle, des techniques d'optimisation et des détails de mise en œuvre de bas niveau. Pour répondre à ces complexités, la synthèse de haut niveau est apparue comme une alternative puissante, en particulier dans la simulation HIL et le développement basé sur FPGA [27–30].

L'HLS permet aux concepteurs de spécifier le comportement des systèmes numériques à l'aide de langages de programmation de haut niveau tels que C, C++ ou SystemC. Contrairement

à la conception RTL traditionnelle, qui nécessite une spécification manuelle du matériel à un faible niveau d'abstraction, l'HLS permet aux concepteurs de se concentrer sur la fonctionnalité au niveau système tout en automatisant la synthèse des composants matériels de bas niveau [31]. Cette abstraction améliore la productivité, accélère les itérations de conception et facilite la validation et la vérification précoces des systèmes numériques complexes. De plus, les outils HLS intègrent des techniques d'optimisation sophistiquées, notamment le déroulage de boucles, le pipelining et la parallélisation, pour améliorer les performances et l'utilisation des ressources [32].

Un avantage clé de l'HLS est sa capacité à générer du matériel compatible FPGA à partir de descriptions de haut niveau, réduisant ainsi considérablement le temps et l'effort de développement [33, 34]. Ce processus est particulièrement bénéfique pour les applications de simulation en temps réel et de prototypage rapide, où la flexibilité de conception et les itérations rapides sont essentielles. Une étude approfondie sur les avantages de l'HLS pour les simulateurs en temps réel basés sur FPGA est présentée dans [35], démontrant les gains d'efficacité et les optimisations de ressources matérielles obtenus grâce aux méthodologies de conception basées sur l'HLS.

Les outils HLS fournissent des pragmas, qui agissent comme des directives de synthèse pour guider et optimiser l'implémentation matérielle [32]. Ces pragmas peuvent être intégrés dans le code source pour influencer des fonctions, des boucles ou des structures de mémoire spécifiques, permettant un contrôle précis de l'architecture matérielle générée. Les avancées récentes en HLS ont conduit au développement de plusieurs outils industriels automatisant la traduction de code C/C++ de haut niveau en représentations RTL optimisées, spécialement adaptées à l'implémentation sur FPGA. Parmi les outils les plus notables figurent Xilinx Vivado HLS, Intel Quartus Prime et Mentor Graphics Catapult C, chacun offrant des capacités robustes de synthèse et de vérification pour rationaliser le processus de conception matérielle. La figure 3.2 illustre le flux de travail d'export RTL utilisant Xilinx Vivado HLS.

En comblant l'écart entre les méthodologies de conception logicielles et l'implémentation matérielle, l'HLS continue de transformer le développement des systèmes numériques, le rendant plus accessible, efficace et adaptable aux exigences technologiques en évolution.

3.6 Représentation des nombres réels sur FPGA

Dans les calculs arithmétiques sur FPGA, le choix de la représentation numérique influence de manière significative les performances du système, l'utilisation des ressources, la latence et la précision. Cette section présente un aperçu comparatif des formats numériques couramment

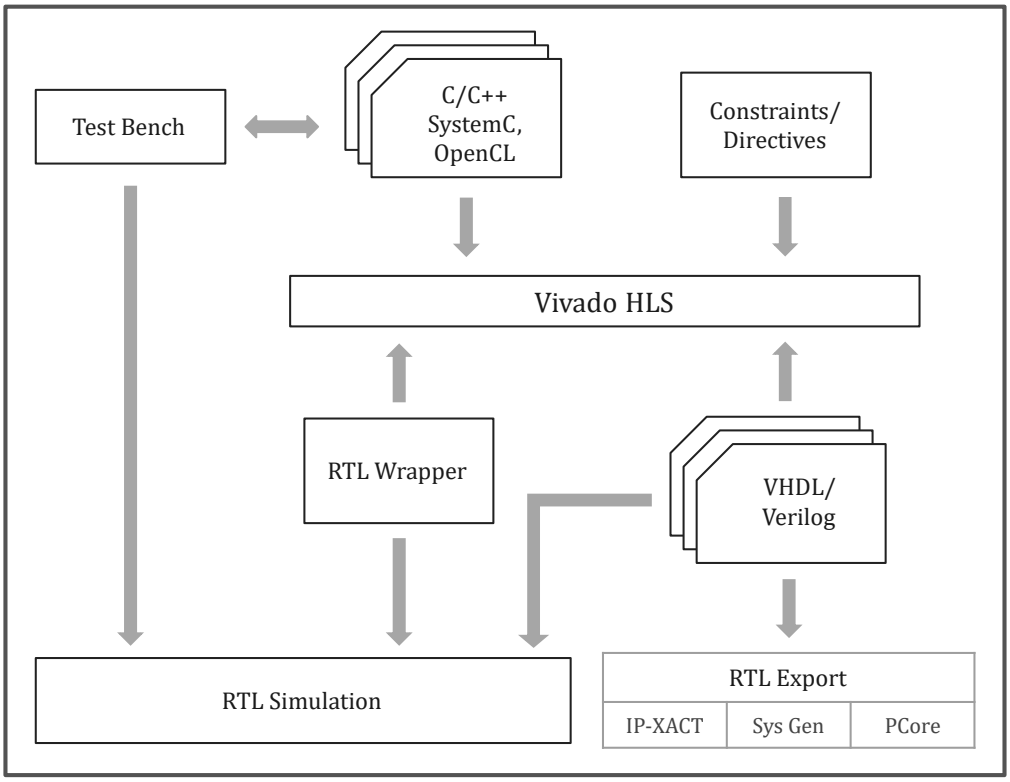


FIGURE 3.2 Flux de conception avec Vivado HLS (inspiré de [36]).

utilisés, virgule fixe, virgule flottante standard et virgule flottante personnalisée, suivi d'une discussion sur la manière dont des formats et bibliothèques adaptés peuvent être exploités dans le cadre d'applications de simulation en temps réel.

3.6.1 Virgule flottante standard

La norme IEEE-754 pour l'arithmétique à virgule flottante est largement utilisée et reconnue pour la représentation des nombres réels à virgule flottante [37]. Elle se compose de trois éléments principaux : le bit de signe, l'exposant et la mantisse. La valeur d'un nombre à virgule flottante est calculée selon la formule suivante :

$$\text{Valeur} = (-1)^{\text{s}} \times 2^{\text{e}} \times \text{m} \tag{3.11}$$

où s indique le signe du nombre, avec 0 pour un nombre positif et 1 pour un nombre négatif, e est l'exposant, généralement stocké sous forme biaisée, ce qui permet de faire varier la valeur par puissances de deux et d'obtenir ainsi une large plage dynamique, et m est la mantisse normalisée, appartenant à l'intervalle [1,2). La mantisse représente les chiffres significatifs du

nombre, fournissant la précision nécessaire. La Figure 6.1 illustre la représentation binaire d'un nombre à virgule flottante codé sur Wfp bits. Le bit de signe occupe un bit, l'exposant biaisé est codé sur We bits, et la fraction (ou partie fractionnaire de la mantisse) est codée sur Wf bits, ce qui donne Wfp = 1 + We + Wf.

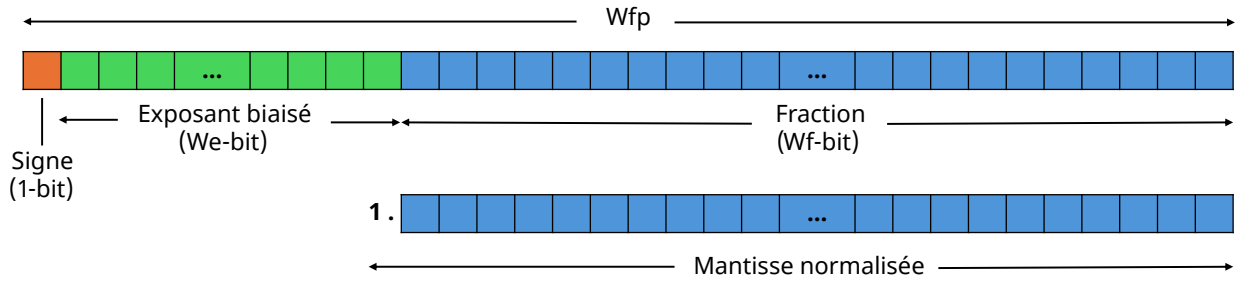


FIGURE 3.3 Structure de la norme IEEE-754 pour la représentation à virgule flottante.

La norme IEEE-754 définit plusieurs niveaux de précision, notamment la demi-précision (16 bits), la simple précision (32 bits), la double précision (64 bits) et la quadruple précision (128 bits), chacun ayant des largeurs de mot différentes, comme présenté dans le Tableau 6.1. Ce codage permet de représenter une large gamme de valeurs, des plus petites aux plus grandes, ainsi que des cas particuliers tels que zéro, l'infini et le NaN (Not-a-Number). Un autre aspect clé du format IEEE-754 est la normalisation : à l'exception des nombres subnormaux, les nombres à virgule flottante sont stockés avec un 1 implicite avant la virgule binaire dans la mantisse, ce qui augmente la précision sans nécessiter de bits supplémentaires.

La norme spécifie également les règles pour l'arrondi, le dépassement (overflow), le sous-dépassement (underflow) et la gestion des exceptions, assurant un comportement numérique cohérent et prévisible sur différentes plateformes matérielles et logicielles. Grâce à sa flexibilité et à sa grande plage dynamique, l'arithmétique IEEE-754 est largement utilisée dans les domaines du calcul scientifique, du traitement graphique et des applications nécessitant une grande précision numérique. Toutefois, cette flexibilité implique une complexité matérielle plus élevée et une consommation accrue de ressources lorsqu'elle est implémentée sur FPGA.

TABLEAU 3.1 Spécifications du format à virgule flottante.

Format	**Taille (Wfp)**	**Exposant (We)**	**Fraction (Wf)**	**Biais**
Demi-précision	16	5	10	15
Single-precision	32	8	23	127
Double-precision	64	11	52	1023
Quadruple-precision	128	15	112	16 383

L'augmentation du nombre de bits alloués à l'exposant accroît la plage dynamique des valeurs représentables, permettant ainsi de couvrir une gamme plus étendue de grandeurs, depuis des valeurs extrêmement petites jusqu'à des valeurs très élevées. À l'inverse, l'allocation d'un plus grand nombre de bits à la mantisse améliore la précision, car une fraction plus large permet de représenter les nombres avec une granularité plus fine.

3.6.2 Format à virgule fixe

La représentation en virgule fixe est un format numérique alternatif couramment utilisé dans les systèmes embarqués et les implémentations matérielles, où les contraintes de ressources — telles que la logique, la puissance et la mémoire limitées — rendent l'arithmétique en virgule flottante peu pratique. Contrairement aux nombres en virgule flottante, qui séparent l'exposant et la mantisse afin d'offrir une large plage dynamique, les nombres en virgule fixe allouent tous les bits à la représentation d'une valeur avec un facteur d'échelle fixe, échangeant ainsi la plage dynamique contre une simplicité de mise en œuvre et un comportement déterministe.

Un nombre en virgule fixe peut être exprimé comme suit :

$$\text{Valeur} = I + F = \sum_{i=0}^{W_i-1} b_i 2^i + \sum_{j=1}^{W_f} b_{-j} 2^{-j} \tag{3.12}$$

W_i et W_f désignent respectivement le nombre de bits alloués à la partie entière et à la partie fractionnaire, et b_k est la valeur binaire à la position k. La taille totale du mot est donnée par $W_{\text{fxp}} = W_i + W_f$, et un bit supplémentaire est utilisé pour représenter le signe dans les formats signés.

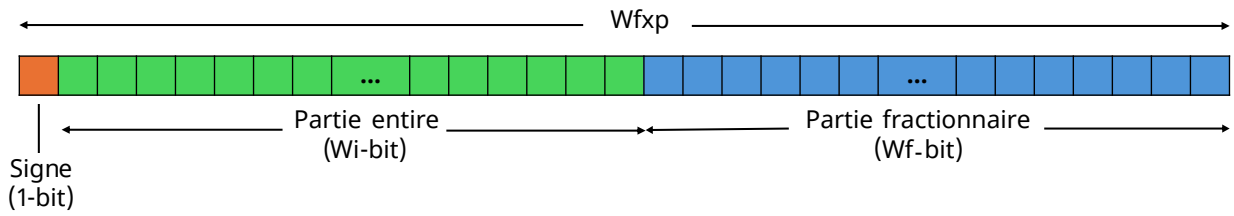


FIGURE 3.4 Structure d'une représentation en virgule fixe signée.

L'arithmétique en virgule fixe présente des avantages significatifs dans les conceptions sur FPGA et ASIC, notamment en raison de son faible coût matériel, de sa latence déterministe et de son utilisation efficace des ressources telles que les blocs DSP et les éléments logiques [38]. Contrairement aux unités en virgule flottante, qui nécessitent des opérations complexes de normalisation et d'arrondi, les unités en virgule fixe s'appuient sur des additionneurs, des décalages et des multiplicateurs entiers, ce qui les rend particulièrement adaptées aux domaines tels que le traitement du signal numérique, les systèmes de contrôle et la simulation en temps réel.

Cependant, cette représentation présente certaines limitations. Les formats en virgule fixe offrent une plage dynamique restreinte et sont sensibles aux phénomènes de débordement et de sous-dépassement. Par ailleurs, des stratégies rigoureuses de mise à l'échelle et de quantification doivent être mises en œuvre pour garantir la précision numérique, en particulier lors d'opérations de multiplication ou de division, où les résultats intermédiaires peuvent excéder la plage représentable. Pour atténuer ces contraintes, les concepteurs réalisent souvent une analyse de précision et utilisent des outils de simulation ou des méthodes analytiques afin de déterminer les tailles de mot optimales ainsi que l'emplacement du point binaire. Lorsqu'elle est soigneusement conçue, l'arithmétique en virgule fixe offre un excellent compromis entre performance et consommation de ressources, ce qui en fait une solution idéale pour les applications à haut débit et à faible consommation, où le coût de la virgule flottante est prohibitif.

3.6.3 Virgule flottante personnalisée

Bien que les formats standard en virgule flottante et en virgule fixe soient largement adoptés, de nombreuses situations, notamment dans les systèmes embarqués, les applications en temps réel et les conceptions FPGA/ASIC, présentent des contraintes strictes en termes de largeur de bits, de consommation d'énergie et de performances. Dans ces contextes, les formats numériques standard peuvent s'avérer sous-optimaux. Pour répondre à ces exigences

spécifiques, des formats numériques personnalisés sont souvent introduits afin d'atteindre un équilibre optimal entre plage dynamique, précision, utilisation des ressources matérielles et latence [39, 40]. Les formats en virgule flottante personnalisés permettent une personnalisation de la largeur des bits, où le nombre de bits alloués au signe, à l'exposant et à la mantisse peut être adapté aux besoins spécifiques d'une application. Par exemple, lorsqu'une large plage dynamique est requise mais qu'une précision modérée suffit, un concepteur peut augmenter la largeur de l'exposant et réduire la taille de la mantisse. Inversement, lorsque la haute précision dans une plage limitée est plus importante, davantage de bits peuvent être attribués à la mantisse tandis que l'exposant est minimisé. Ce niveau de configurabilité permet aux concepteurs de parvenir à un compromis entre la flexibilité de l'arithmétique en virgule flottante, qui gère de grandes plages dynamiques, et l'efficacité des formats en virgule fixe, qui offrent une latence plus faible et des implémentations matérielles plus simples.

Des travaux de recherche récents ont exploré l'utilisation de formats numériques personnalisés dans les bibliothèques RTL et HLS pour répondre à la demande croissante d'efficacité matérielle spécifique au domaine. VFLOAT [41] est une bibliothèque modulaire de cœurs en virgule flottante à précision variable prenant en charge les opérations arithmétiques fondamentales (addition, soustraction, multiplication et division) et la conversion de format entre les représentations en virgule flottante et en virgule fixe. Bien qu'elle serve de référence pour la validation et le test des unités en virgule flottante, son objectif principal est la vérification fonctionnelle plutôt que le déploiement en production.

FloPoCo [42, 43], abréviation de "FLoating Point Operator COres", est un générateur open-source de cœurs arithmétiques FPGA qui produit automatiquement des implémentations VHDL/Verilog optimisées et personnalisables pour des opérations allant de l'arithmétique de base aux fonctions transcendantes, prenant en charge les formats en virgule fixe, en virgule flottante, entiers et complexes. Contrairement aux outils commerciaux, il génère du RTL lisible par l'homme avec des optimisations conscientes de l'architecture pour une efficacité supérieure en termes de surface et de performances dans les implémentations FPGA. Les opérateurs à précision évolutive du framework le rendent particulièrement précieux pour les applications à ressources limitées nécessitant une précision numérique adaptée.

Malgré l'utilisation de longue date des implémentations en virgule flottante dans les FPGA, l'intégration de formats en virgule flottante personnalisés dans les méthodologies de synthèse à haut niveau reste limitée. Cette limitation peut être attribuée à plusieurs facteurs, notamment l'accent prédominant de nombreux outils HLS sur l'arithmétique en virgule fixe et la complexité inhérente à l'équilibre entre performances et utilisation efficace des ressources dans les conceptions en virgule flottante. Bien que diverses approches aient été proposées pour

répondre au besoin de bibliothèques en virgule flottante “soft” dans les cadres HLS [44], ces bibliothèques manquent souvent de la capacité à générer des cœurs IP paramétrés au moment de la compilation. En conséquence, elles sont généralement limitées à un ensemble fixe de largeurs d'opérateurs, ce qui limite leur flexibilité et leur adaptabilité à diverses applications.

Une direction notable implique l'utilisation de HLS pour implémenter des opérations en virgule flottante non standard, telles que des unités de sommation personnalisées [45]. Cependant, de telles approches ne prennent généralement pas en charge la précision hétérogène, restreignant ainsi les avantages potentiels de l'adaptation de la précision à différentes parties de la conception. Par conséquent, il reste une opportunité significative d'avancer les méthodologies HLS pour prendre en charge nativement la génération d'unités arithmétiques en virgule flottante personnalisables, efficaces et évolutives qui répondent à la fois aux exigences spécifiques des applications et aux contraintes matérielles.

Template HLS (THLS) [46] aborde la mise en œuvre d'opérations en virgule flottante personnalisées à un niveau élevé grâce à l'utilisation de modèles C++. Il permet la spécification à la compilation des largeurs d'exposant et de fraction et prend en charge les configurations de précision mixte pour les arguments d'entrée et les types de résultats [47]. Bien que THLS fournisse un cadre unifié pour la simulation et la synthèse, sa portée reste largement limitée aux opérations fondamentales, laissant des opportunités pour une optimisation supplémentaire et l'inclusion d'opérations supplémentaires spécifiques à l'application.

TrueFloat [48], intégré dans l'outil HLS open-source Bambu [49], représente une autre avancée significative dans ce domaine. Cette intégration introduit de nouvelles opportunités d'optimisation et permet la génération de représentations en virgule flottante équivalentes à un niveau d'abstraction plus élevé. TrueFloat facilite la traduction entre différents formats en virgule flottante grâce à une interface simplifiée basée sur des options de ligne de commande simples.

Une autre contribution pertinente est l'architecture Fused Vector FP [50]. Dans ce travail, les auteurs présentent une architecture de produit scalaire en virgule flottante à plusieurs termes fusionnés, paramétrée, adaptée à la synthèse à haut niveau. Cette approche combine l'efficacité computationnelle d'une conception de produit scalaire fusionné avec la flexibilité de HLS, permettant une optimisation fine pour les performances et l'efficacité des ressources à travers divers formats de données et configurations matérielles. Malgré ses résultats prometteurs, l'architecture présente certaines limitations. Notamment, elle ne prend pas en charge les formats hétérogènes, réduisant ainsi son adaptabilité à un large éventail de scénarios computationnels. De plus, contrairement aux approches précédemment mentionnées orientées FPGA, l'architecture Fused Vector FP cible les implémentations ASIC.

3.7 Approches de l'état de l'art

La simulation des transitoires électromagnétiques en temps réel joue un rôle essentiel dans le développement des systèmes électriques. Cependant, avec l'évolution technologique et le besoin croissant de fréquences de commutation élevées dans les systèmes modernes, les approches de simulation traditionnelles peuvent ne pas suffire à répondre aux exigences actuelles. Ces dernières années, des efforts significatifs ont été déployés pour pallier ces limites. Les chercheurs et les professionnels de l'industrie ont reconnu la nécessité de techniques de simulation plus efficaces et plus précises pour suivre le rythme des exigences évolutives des réseaux électriques. Dans ce contexte, les FPGAs ont été exploités pour permettre des simulations plus rapides et plus précises, en particulier pour les systèmes à haute fréquence de commutation et à comportements dynamiques complexes.

Dans [51], un système de simulation EMT basé sur FPGA est proposé pour modéliser les dispositifs haute fréquence. En exploitant les capacités de traitement parallèle des FPGAs, les algorithmes EMT peuvent être parallélisés afin de réduire le pas de temps et d'améliorer l'efficacité de la simulation. Par ailleurs, l'article [52] présente un simulateur EMTP en temps réel basé sur FPGA, dans lequel le modèle de ligne à dépendance fréquentielle (FD-line) joue un rôle central pour permettre le calcul précis des transitoires de ligne. L'emploi de modèles plus précis dans la simulation ajoute inévitablement une surcharge de calcul, ce qui est en contradiction avec les contraintes du temps réel.

Pour répondre à ce défi, diverses approches visant à accroître le parallélisme ont été explorées. L'article [53] propose différents niveaux de parallélisme pour réduire la charge de calcul. L'idée principale est d'assurer la reconfigurabilité afin d'adapter la plateforme à différentes topologies de réseaux électriques. L'approche de découplage et l'utilisation d'une représentation en virgule fixe permettent également de réduire le temps de calcul. Le travail [21] traite des principaux défis liés à l'implémentation EMTP sur FPGA, notamment la complexité de conception des modules matériels et de leur interconnexion tout en assurant un haut niveau de parallélisme. Ce travail propose une stratégie de partage de chemins de calcul ainsi que des algorithmes de parallélisation pour améliorer l'implémentation EMTP sur FPGA. Toutefois, des améliorations supplémentaires peuvent être obtenues en exploitant davantage le parallélisme intrinsèque du FPGA et en explorant des optimisations logicielles et matérielles supplémentaires.

La méthode MNA est une technique numérique largement utilisée pour résoudre les équations différentielles régissant le comportement des circuits électriques. Elle transforme les équations du circuit, généralement formulées sous forme nodale, en un système matriciel creux. Cepen-

dant, à mesure que la complexité du circuit augmente, la croissance de la sparsité de la matrice pose un défi majeur pour l'efficacité du calcul.

Des recherches récentes ont visé à optimiser l'implémentation de la MNA pour remédier à ces problèmes d'évolutivité. Par exemple, [54] propose des stratégies efficaces pour la gestion des grandes matrices creuses, en mettant l'accent sur la vitesse de calcul et l'efficacité mémoire. L'article [9] propose un simulateur en temps réel pour les systèmes électriques basé sur la méthode MANA et son implémentation HIL. Ce travail utilise également les FPGAs pour exploiter leur parallélisme intrinsèque dans le but de réduire le pas de temps de simulation des convertisseurs de puissance. Bien que cette approche soit bien adaptée aux convertisseurs à faible ou moyenne fréquence de commutation, l'article [55] aborde les défis spécifiques liés aux convertisseurs à haute fréquence de commutation. Il propose des méthodes de modélisation générales permettant d'obtenir une simulation en temps réel précise de ces convertisseurs. L'objectif principal est le convertisseur de puissance, mais les stratégies proposées peuvent être généralisées à d'autres applications traitant des défis de simulation en temps réel à haute fréquence de commutation.

Dans [56], une plateforme de simulation en temps réel basée sur FPGA est proposée pour les convertisseurs électroniques de puissance, dans le but de fournir une méthode plus rapide et plus efficace pour tester et améliorer les algorithmes de commande par rapport aux simulations logicielles traditionnelles. La plateforme utilise un processeur basé sur FPGA et consiste à traduire les équations du convertisseur, modélisées en langage C, en langage assembleur pour le processeur. Toutefois, plusieurs limitations ont été identifiées, notamment la difficulté de passer à des fréquences de commutation nettement plus élevées, les contraintes potentielles de précision liées à l'utilisation probable d'une arithmétique en virgule fixe avec une résolution de 12 bits, la complexité de la programmation en assembleur et les problèmes d'évolutivité, ainsi qu'une latence reconnue du système.

L'article [57] présente un nouveau modèle de ligne dépendant de la fréquence (FD-Line) pour la simulation numérique en temps réel des systèmes électriques. Le modèle FD-Line a été intégré à l'environnement Simulink et utilisé pour des simulations EMT hors ligne à l'aide de la boîte à outils PSB/Simulink. Les auteurs ont également mis en œuvre leur méthode sur des simulateurs en temps réel grâce à la parallélisation. Bien que des pas de temps réduits aient été atteints, les résultats n'étaient pas encore satisfaisants pour les systèmes larges et complexes, ce qui a motivé la poursuite des recherches pour améliorer les performances de simulation.

Dans [58], une méthodologie de conception pour la simulation du Universal Line Model (ULM) sur FPGA est proposée. L'ULM est un modèle en domaine de phase dépendant de la

fréquence pour les lignes et câbles de transmission. Une approche basée sur l'espace d'état est utilisée pour simuler dans le domaine temporel la forme pôle-résidu de l'ajustement rationnel de l'admittance caractéristique et des fonctions de propagation. Ce travail explore également la planification des calculs ULM et la gestion des termes historiques afin d'optimiser l'utilisation du matériel et de réduire la latence. Un aspect intéressant est l'utilisation d'un format en virgule flottante non standard permettant de trouver un compromis entre l'utilisation des ressources et la précision des calculs.

L'article [59] vise à améliorer le processus de conception et la flexibilité de l'implémentation des algorithmes EMT en tirant parti des techniques de HLS. Il explore le parallélisme potentiel offert par le HLS et propose un cadre de simulation simple pour démontrer ses capacités dans le contexte de la simulation en temps réel. Toutefois, la principale limite de ce travail réside dans la simplicité du cadre utilisé, qui ne permet pas une exploitation complète du potentiel offert par le HLS. Des travaux supplémentaires sont nécessaires pour évaluer pleinement les avantages du HLS dans des scénarios de simulation plus complexes.

L'article [60] propose une méthode de partitionnement pour réduire la charge de calcul du modèle FDNE en exploitant la sparsité des matrices d'état et en parallélisant les calculs. En utilisant une conception HLS et une implémentation sur FPGA, une simulation en temps réel avec des pas de temps de l'ordre de la microseconde est obtenue. Ce travail se concentre principalement sur la parallélisation du modèle FDNE, sans approfondir l'exploitation des fonctionnalités parallèles intrinsèques du FPGA.

Malgré les avancées significatives réalisées dans [60] pour améliorer la performance des modèles FDNE—telles que la réduction de la charge de calcul et l'obtention de pas de temps plus petits—certaines applications, notamment celles impliquant des systèmes complexes et de grande taille avec des convertisseurs à haute fréquence de commutation, nécessitent encore des améliorations supplémentaires pour atteindre des pas de temps de l'ordre de la sous-microseconde.

3.8 Conclusion

Cette revue de littérature a fourni un aperçu des divers aspects liés à notre sujet de recherche. Nous avons présenté les différentes approches et considérations pour la conception et la mise en œuvre de simulateurs en temps réel pour les réseaux électriques, avec un accent particulier sur les plateformes à base de FPGA. La revue a mis en évidence les limites des simulateurs traditionnels basés sur des processeurs CPU, qui, bien que flexibles, peinent à satisfaire les exigences de latence et de parallélisme requises par les simulations de transitoires électroma-

gnétiques. En revanche, les approches basées sur FPGA offrent une exécution déterministe et une capacité de traitement à haut débit, ce qui les rend particulièrement adaptées à ces applications critiques en temps.

Une attention particulière a été portée à la modélisation et à la simulation des modèles FDNE, qui permettent d'approcher de manière fidèle les effets dynamiques du réseau externe dans les études EMT. Plusieurs méthodes de réalisation des FDNE ont été examinées, notamment l'approximation par fonctions rationnelles, la formulation en espace d'états et l'implémentation en temps discret. L'intégration de ces modèles FDNE dans des architectures FPGA soulève des défis en matière de stabilité numérique, d'utilisation des ressources et de latence, qui ne sont que partiellement abordés dans les travaux existants.

Par ailleurs, nous avons analysé l'impact de la résolution numérique sur les performances de simulation. Les représentations en virgule fixe et en virgule flottante selon la norme IEEE-754 présentent des compromis distincts en termes de précision, de plage dynamique et d'efficacité matérielle. Bien que largement utilisées, les formats standards s'avèrent souvent insuffisants pour équilibrer performances et contraintes de ressources. Cette limite a suscité un intérêt croissant pour les formats à virgule flottante personnalisés, adaptables aux besoins spécifiques des applications. Toutefois, notre étude a révélé que les outils HLS actuels offrent un soutien limité pour de telles personnalisations, et que peu de solutions permettent une configurabilité à la compilation ou le support d'une arithmétique hétérogène.

En réponse à ces lacunes, les travaux de recherche présentés dans cette thèse visent à développer une méthodologie complète pour la simulation en temps réel des FDNE et des systèmes électroniques de puissance à l'aide de plateformes FPGA. L'accent est mis sur l'utilisation de formats numériques personnalisables et d'unités arithmétiques optimisées afin d'améliorer l'efficacité du calcul sans compromettre la précision. Les chapitres suivants présentent les contributions originales de cette recherche, s'appuyant sur les enseignements de cette revue pour proposer de nouvelles stratégies de conception, des bibliothèques de haut niveau et des validations expérimentales visant à faire progresser l'état de l'art dans ce domaine.

CHAPITRE 4 ORGANIZATION DU CONTENU ET APPROCHE DE LA RECHERCHE

Cette thèse adopte un format par articles afin de présenter les travaux de recherche réalisés au cours de ce programme doctoral. Les cinq chapitres suivants exposent les contributions principales, structurées autour de cinq articles évalués par des pairs et publiés. Ces articles introduisent et analysent collectivement la méthodologie développée pour la mise en œuvre de simulateurs en temps réel sur des FPGA. Chaque article représente une étape progressive dans la formulation de l'approche proposée, avec des résultats complémentaires contribuant à une méthodologie cohérente. L'organisation de ces travaux suit un ordre chronologique.

1. Le premier article discute la simulation en temps réel basée sur FPGA des FDNE. Dans cette publication, un système électrique d'avion est utilisé comme cas d'étude. Diverses représentations numériques, pour une mise en œuvre sur FPGA y sont examinées, notamment les arithmétiques en virgule flottante double précision, en virgule flottante simple précision et en virgule fixe.
2. Le second article introduit une nouvelle bibliothèque HLS, CuFP, conçue pour une représentation en virgule flottante personnalisée, offrant une alternative flexible qui équilibre les avantages des arithmétiques en virgule fixe et en virgule flottante.
3. Le troisième article améliore CuFP en intégrant un format d'auto-alignement (Self-Alignment Format, SAF). Cette amélioration optimise le chemin de données en réduisant la latence et la consommation de ressources grâce à un alignement stratégique des opérandes et à l'élimination des étapes de traitement inutiles.
4. Le quatrième article explore des stratégies alternatives pour modéliser les FDNE, en introduisant et comparant deux approches distinctes en termes d'efficacité de mise en œuvre et de fidélité de simulation. Un STATCOM est utilisé comme cas d'étude. De plus, la publication démontre comment les représentations numériques et les unités arithmétiques personnalisées développées précédemment peuvent être appliquées à la simulation sur FPGA de convertisseurs connectés au réseau, mettant en évidence à la fois la précision et l'efficacité.
5. Le cinquième article porte sur les additionneurs à retenue sauvegardée à base élevée (High-Radix Carry-Save adder, HRCS). Étant donné que l'addition est l'une des opérations les plus fréquemment utilisées dans le calcul en temps réel et les applications basées sur FPGA, cette contribution vise à optimiser davantage la performance et l'efficacité spatiale des unités arithmétiques utilisées dans les cadres de simulation.

Les recherches existantes démontrent que les simulateurs en temps réel basés sur FPGA peuvent prendre en charge la mise en œuvre de solveurs avec des pas de temps très petits. Cependant, de nombreuses implémentations reposent sur des conceptions RTL personnalisées, qui offrent une flexibilité limitée pour la reconfiguration. La complexité associée à la conception RTL nécessite les connaissances de spécialistes du matériel et limite l'intervention des spécialistes des applications ou des développeurs de logiciels, ce qui est présenté comme un inconvénient. En revanche, les approches HLS offrent une alternative plus conviviale, rendant le développement FPGA plus accessible aux spécialistes des applications et aux développeurs de logiciels.

Le premier article introduit une approche basée sur les pôles et résidus pour modéliser les FDNE, mise en œuvre à l'aide d'outils HLS. L'étude évalue plusieurs représentations numériques, notamment les arithmétiques en virgule flottante double précision, en virgule flottante simple précision et en virgule fixe, dans le contexte de la mise en œuvre sur FPGA. Les résultats démontrent que l'approche proposée peut atteindre des pas de temps de simulation inférieurs à la microseconde. Les expériences ont été initialement menées sur un système de test à petite échelle pour évaluer la performance des différents types de données. Les résultats montrent que, bien que l'arithmétique en virgule flottante double précision offre la plus grande précision numérique, elle entraîne également une consommation de ressources et une latence de calcul significativement plus élevées. L'arithmétique en virgule flottante simple précision offre un compromis, avec une précision inférieure mais une latence et une efficacité des ressources améliorées par rapport à la double précision. En revanche, la représentation en virgule fixe permet de personnaliser la largeur des bits pour les parties entière et fractionnaire, permettant une optimisation basée sur les exigences spécifiques de l'application. Dans cette étude, la mise en œuvre en virgule fixe a démontré une latence inférieure à celle de la simple précision, la rendant particulièrement bien adaptée à la simulation en temps réel de systèmes plus grands. Elle a également consommé moins de ressources tout en maintenant un niveau d'erreur numérique comparable à celui de la simple précision. Par conséquent, le format en virgule fixe a été sélectionné pour les étapes expérimentales ultérieures présentées dans cet article.

S'appuyant sur les enseignements du premier article, le deuxième article aborde les limitations des résolutions numériques existantes utilisées dans les simulateurs en temps réel basés sur FPGA. Une observation critique dans la littérature est l'utilisation prédominante des représentations en virgule flottante standard IEEE dans ces simulateurs. Bien que l'arithmétique en virgule flottante fournisse une large plage dynamique, essentielle pour modéliser avec précision les systèmes électriques avec de grandes variations de tension ou de courant, elle entraîne une pénalité significative en termes d'utilisation des ressources et de latence de

calcul lorsqu'elle est mise en œuvre sur des plateformes FPGA. En revanche, l'arithmétique en virgule fixe est plus efficace en termes de ressources et de latence, mais manque de la plage dynamique nécessaire dans de nombreux scénarios pratiques. Néanmoins, les formats en virgule fixe offrent un certain degré de flexibilité en permettant aux développeurs d'ajuster le nombre de bits alloués aux parties entière et fractionnaire, les rendant potentiellement plus adaptés aux applications en temps réel, en particulier dans les systèmes à grande échelle où les contraintes de calcul sont plus critiques.

Pour combler cet écart entre performance et configurabilité, le deuxième article introduit une nouvelle bibliothèque HLS pour les opérateurs en virgule flottante personnalisés, nommée CuFP (Customized Floating-Point library). Cette bibliothèque offre la possibilité de définir des formats en virgule flottante adaptés aux exigences spécifiques de précision et de plage de l'application cible. En permettant une personnalisation au niveau du bit de l'exposant et de la mantisse, CuFP permet des compromis entre plage dynamique, précision, latence et utilisation des ressources. Cette approche combine efficacement les avantages des représentations en virgule flottante et en virgule fixe : elle conserve la plage dynamique de l'arithmétique en virgule flottante tout en offrant la configurabilité et l'efficacité généralement associées aux conceptions en virgule fixe.

La bibliothèque est conçue pour être hautement portable et s'intègre parfaitement dans les flux de conception HLS, la rendant accessible aux ingénieurs d'application sans besoin d'expertise en conception matérielle de bas niveau. Plusieurs configurations sont évaluées pour démontrer la flexibilité de CuFP à équilibrer la précision numérique avec les contraintes matérielles. Ces expériences confirment que les formats en virgule flottante personnalisés peuvent réduire significativement la latence et l'utilisation des ressources des simulateurs basés sur FPGA tout en maintenant la précision requise pour une simulation en temps réel stable et fiable.

Poursuivant ces résultats, le troisième article améliore la bibliothèque en intégrant la technique d'auto-alignement, conduisant au développement de la bibliothèque CuFPSAF. Cette intégration fournit des opérations en virgule flottante personnalisées liées à l'addition avec une précision et une utilisation des ressources améliorées, et réalise des réductions significatives de la latence de calcul et de la surface matérielle en rationalisant l'alignement et en supprimant les étapes inutiles. Cette technique garantit que les décalages fins de la mantisse, intensifs en calcul, sont déplacés en dehors du chemin critique, découplant efficacement la surcharge des ajustements de la mantisse du processus d'addition principal. Cette approche rationalisée contribue directement à des réductions significatives de la latence de calcul et permet des opérations arithmétiques plus rapides et plus fiables. Le SAF lui-même est conçu

pour améliorer la précision en étendant la mantisse, en incorporant le signe, en transférant une partie de l'exposant et en ajoutant des bits de garde. De plus, le SAF réalise des réductions substantielles de la surface matérielle en préservant l'alignement de la mantisse et de l'exposant et en éliminant les étapes de conditionnement et de déconditionnement inutiles du chemin de données critique. La bibliothèque CuFPSAF, construite sur cette intégration à l'aide d'une architecture C++ générique fondée sur l'utilisation de templates, permet une personnalisation flexible des paramètres et prend en charge différentes approches (contraintes de latence et de ressources) pour équilibrer la performance et l'utilisation des ressources, la rendant bien adaptée au déploiement efficace sur FPGA d'opérateurs en virgule flottante complexes pour diverses applications.

Pour évaluer l'efficacité des représentations numériques et des bibliothèques arithmétiques personnalisées développées précédemment dans un contexte plus complexe et réaliste, le quatrième article présente un cadre de simulation en temps réel basé sur FPGA pour un compensateur synchrone statique (Static Synchronous Compensator, STATCOM) connecté au réseau. Ce cas test aborde les défis croissants posés par l'intégration accrue des convertisseurs électroniques de puissance dans les systèmes électriques modernes. S'appuyant sur les contributions antérieures, le travail applique et évalue les bibliothèques CuFP et CuFPSAF dans un contexte de simulation détaillé, démontrant leur rôle dans la réalisation d'opérations arithmétiques efficaces et précises sur FPGA. Une contribution clé de cet article est l'introduction et la comparaison de deux approches de modélisation distinctes pour les FDNE, chacune offrant des compromis en termes de latence, d'utilisation des ressources et de fidélité de simulation. Le STATCOM sert de cas test représentatif pour intégrer ces modèles FDNE dans un cadre HIL, facilitant des tests en temps réel sûrs et contrôlés. Les composants STATCOM et FDNE sont formulés en représentation d'état et discrétisés à l'aide de la méthode d'Euler arrière (Backward Euler) pour assurer la stabilité numérique et une computation matricielle efficace. Une validation expérimentale par rapport à un modèle de référence EMTP confirme la précision des résultats de simulation. L'implémentation sur FPGA avec la carte Alveo U280, en utilisant des outils HLS, permet d'atteindre un haut débit et une grande efficacité des ressources, soulignant la pertinence de l'approche proposée pour les applications temps réel dans les systèmes électriques.

Poursuivant la trajectoire de développement de cadres arithmétiques optimisés pour la simulation temps réel sur FPGA, le cinquième article s'attaque à la problématique de la sommation efficace de multiples entrées sur des plateformes FPGA, une opération essentielle dans les moteurs de simulation, notamment pour la résolution d'équations matricielles et les réductions de vecteurs. Alors que les travaux précédents se concentraient sur l'arithmétique à virgule flottante personnalisée et l'optimisation de l'alignement, cet article se focalise sur l'addition,

l'une des opérations les plus fréquemment exécutées dans le calcul temps réel. Les additionneurs classiques, tels que ceux à propagation de retenue (ripple-carry), souffrent de goulots d'étranglement en termes de latence, ce qui peut compromettre les performances de simulation à haute fréquence d'échantillonnage. Pour y remédier, ce cinquième article explore le format HRCS comme solution pour minimiser les délais de propagation de retenue. En représentant les nombres par des composants de somme et de retenue séparés pour chaque chiffre, le format HRCS permet une sommation parallèle sans résolution immédiate de la retenue, ce qui le rend particulièrement adapté à la construction d'arbres d'addition à plusieurs niveaux. L'article propose un additionneur endomorphique au format HRCS, capable de traiter des entrées et de produire des sorties dans ce même format, facilitant ainsi l'enchaînement des opérations et leur intégration dans des chemins de données complexes. S'appuyant sur une version antérieure d'un additionneur HRCS à chaîne de retenue unique, ce travail propose une architecture améliorée intégrant un mécanisme de gestion de la retenue repensé afin de minimiser le délai sur le chemin critique. Cette optimisation repose sur une étroite intégration de la logique de retenue initiale avec une structure dédiée de type LUT. La conception est validée dans le contexte d'une opération de réduction vectorielle, particulièrement sensible à la latence, comme celles utilisées pour la résolution d'équations systèmes en simulation temps réel. Les résultats expérimentaux démontrent des réductions significatives de la latence à mesure que la base (radix) augmente, sans augmentation notable de l'utilisation des ressources. Cette contribution vient compléter les travaux antérieurs en optimisant davantage le chemin de données arithmétique, garantissant que des opérations fondamentales telles que l'addition ne constituent plus un frein aux performances dans les simulateurs temps réel sur FPGA. En intégrant l'additionneur HRCS proposé aux architectures de simulation reposant sur les bibliothèques CuFP et CuFPSAF, le système dans son ensemble bénéficie d'améliorations en termes de latence, de mise à l'échelle et d'efficacité des ressources, des attributs essentiels pour faire progresser les capacités des systèmes HIL temps réel.

CHAPITRE 5 ARTICLE 1 : FPGA-BASED FDNE MODELS FOR THE ACCURATE REAL-TIME SIMULATION OF POWER SYSTEMS IN AIRCRAFTS

Fahimeh Hajizadeh, Loïc Alavoine, Tarek Ould-Bachir, Frédéric Sirois, and Jean-Pierre David

Publié dans : 12th International Conference on Renewable Energy Research and Applications (ICRERA), Oshawa, ON, Canada

Date de parution : 6 octobre 2023

Abstract

The advancements in aircraft technology, particularly the increased electrification of aircraft, have introduced complexities in designing and integrating electrical systems. Hardware-in-the-loop (HIL) simulation is recognized as a crucial tool in the aerospace industry, utilizing Field-Programmable Gate Arrays (FPGAs) for their high-speed data processing and low latency. This paper presents a method for accurate real-time simulation of cable harnesses in modern aircrafts, employing Frequency-Dependent Network Equivalent (FDNE) models and FPGAs. The implementation methodology and results of the FPGA-based HIL simulation are presented and discussed. Various equivalent models with different data types are evaluated to determine achievable time-step and resource consumption. Results show the effectiveness of the proposed method in achieving sub-microsecond time-step.

Keywords

Frequency-Dependent Network Equivalent (FDNE), Real-time simulation, High-level synthesis, FPGA

5.1 Introduction

The advancement of aircraft technology has led to the development of Fly-By-Wire (FBW) and More Electric Aircraft (MEA), driven by the goal of reducing weight and maintenance costs while improving overall performance. It is essential to validate and test conceptual designs during the early stages of electrical system design in order to overcome these challenges. Traditional testing methods in aerospace often involve expensive and high-maintenance test rigs [61]. As an alternative, real-time simulation offers a cost-effective, time-efficient, and

flexible solution for testing new technologies, making it an appealing choice in the aerospace industry.

Hardware-in-the-loop (HIL) simulation has emerged as a crucial tool in the aerospace industry to evaluate the performance, reliability, and safety of these systems [62]. HIL enables the testing of aircraft systems under realistic operating conditions, allowing engineers to assess the system's response and behavior in a controlled and repeatable environment. The round-trip time plays a key role in HIL simulations since it represents the time taken for a signal to travel between the simulated and real hardware components. Minimizing round-trip time is crucial for real-time interaction between simulated and physical components in HIL simulations.

FPGAs (Field Programmable Gate Arrays) are configurable circuits that enable the implementation of custom digital circuits. Because of their low latency capabilities, they have emerged as an interesting solution in HIL simulations [15, 21].

In recent years, efforts have been made to study and simulate aircraft power systems in offline and real-time [63, 64]. In [63], a suitable power system architecture for a future MEA is proposed, with a focus on analyzing system stability. The authors utilize offline simulation to validate the effectiveness of the proposed architecture. In [64], a whole power system of an aircraft is implemented in offline and real-time. Real-time simulation is done using OPAL-RT's platforms and is based on CPUs. In this paper, AC and DC cables are replaced by simple RL equivalent models, and the complex impedance effect is ignored. In this regard, their proposed model suffers from a high level of accuracy. Moreover, the authors have not taken into account more sophisticated systems with high switching frequencies.

To overcome this limitation, this paper presents a novel approach using frequency-dependent network equivalent (FDNE) models to accurately represent aircraft power cables. Recent progress in implementing real-time frequency-dependent (FD) models has been demonstrated in studies [57, 60, 65]. For example, Cortez et al. [57] developed an FD line model for real-time power system simulation on CPUs, focusing on transmission networks with tens of microseconds time-steps. Ould-Bachir et al. [58] proposed an FPGA-based simulation with sub-microsecond time-steps, targeting fault location on transmission lines. However, these works focused on long transmission lines spanning several kilometers, while our focus is on much shorter cables, just a few meters in length. Dicler [60] explored real-time FDNE models on CPUs and FPGAs. He investigated on parallelizing FDNE algorithm and proposed a partitioning algorithm to compute the FDNE. In addition, he implemented the North Brazilian System using the proposed FDNE model. However, the reported time-steps for the systems with different ports and poles are in the microsecond range.

The remainder of this paper is organized as follows : Section 5.2 presents an overview of

FDNE models, outlining the design formulation and describing the proposed method. In Section 5.3, the proposed implementation methodology is explained in detail. The FPGA implementation results for real-time simulation of FDNE models using the accelerated HLS-based approach are discussed in Section 5.4. Finally, Section 5.5 provides a conclusion, and forecasts future developments of FPGA-based real-time FDNE models for aircraft power system testing applications.

5.2 Time-domain Simulation of FDNE Models

An FDNE captures the frequency response of a network at its ports :

$$\mathbf{i} = \mathbf{Y}(s)\mathbf{v} \tag{5.1}$$

where $\mathbf{i}$ and $\mathbf{v}$ are respectively the current and voltages at the ports of the network, and $\mathbf{Y}(s)$ is the frequency-dependent symmetrical admittance matrix of size p, the number of ports in the network :

$$\mathbf{Y}(s) = \begin{bmatrix} y_{11}(s) & \dots & y_{1p}(s) \\ \vdots & \ddots & \vdots \\ y_{p1}(s) & \dots & y_{pp}(s) \end{bmatrix} \tag{5.2}$$

In order to conduct time-domain simulation of the FDNE, $\mathbf{Y}(s)$ can first be approximated using a partial decomposition, as shown in Eq. 7.1.

$$\mathbf{Y}(s) \cong \mathbf{Y}_{fitted}(s) = \mathbf{G}_0 + s\mathbf{E} + \sum_{k=1}^{n} \frac{\mathbf{R}_k}{s - p_k} \tag{5.3}$$

where coefficients p_k and $\mathbf{R}_k$ are poles and residues matrices, respectively ; n denotes the number of poles of the model ; and $\mathbf{G}_0$ and $\mathbf{E}$ are constant matrices ($\mathbf{E}$ is typically a zero matrix). The approximation can be obtained through various techniques, among which vector fitting (VF) [26] and rational Krylov fitting (RKFit) [66] are popular choices.

The time-domain equations of the FDNE are obtained using numerical integration methods, such as the backward Euler method or the trapezoidal rule, and yields :

$$\mathbf{i}(t) = \mathbf{G}_Y\mathbf{v}(t) + \mathbf{i}^h(t) \tag{5.4}$$

where $\mathbf{G}_Y$ is the time-domain admittance matrix, and $\mathbf{i}^h(t)$ is the history term vector. These terms are obtained through from the fixed-step integration scheme. Trapezoidal integration

method is often chosen for its L-stability, as well as its simplicity and accuracy. However, on FPGA, backward Euler (BE) should be preferred for its A-stability in the presence of switching devices in the network, while taking advantage of the small simulation time-steps that compensates its less accurate nature compared to the trapezoidal method.

Using BE, one obtains $\mathbf{G}_Y$ as :

$$\mathbf{G}_Y = \mathbf{G}_0 + \sum_{k=1}^{N_Y=n} \frac{\Delta t \mathbf{R}_k}{1 - p_k \Delta t} \tag{5.5}$$

and $\mathbf{i}^h(t)$ as :

$$\mathbf{i}^h(t) = \sum_{k=1}^{n} \mathbf{x}_k(t) \tag{5.6}$$

where, $\mathbf{x}_k(t)$ are state vectors associated with the poles and residues p_k and $\mathbf{R}_k$:

$$\mathbf{x}_k(t) = \alpha_k \mathbf{x}_k(t - \Delta t) + \boldsymbol{\beta}_k \mathbf{v}(t - \Delta t) \tag{5.7}$$

with

$$\alpha_k = \frac{1}{1 - p_k \Delta t}, \boldsymbol{\beta}_k = \frac{\Delta t \mathbf{R}_k}{(1 - p_k \Delta t)^2} \tag{5.8}$$

5.3 Implementation Methodology

5.3.1 FDNE Hardware Architecture

Fig. 5.1a illustrates a general view of an FDNE model with two ports. This FDNE model interacts with a nodal solver, as shown in Fig.5.2, through a Norton equivalent circuit illustrated in Fig. 5.1b and given by Eq. 5.4. Fig.5.2a indicates the interaction of the nodal solver and the FDNE model. The nodal solver handles network equations using the Modified-Augmented Nodal Analysis (MANA). The FDNE block and nodal solver work serially, and the FDNE model is fed by the MANA output (voltage). The FDNE Module consists of Eqs. 5.6 and 5.7. Fig. 5.2b illustrates a high level inside view of the FDNE module. Accordingly, the FDNE module is composed of a Matrix-Vector Multiplier (MVM), scalar-vector multipliers, and accumulators. Depending on the size of the network, the required number of each components and their input sizes may increase linearly or quadratically. The size of the network is also proportional to the number of poles and ports, and thus the number of system states.

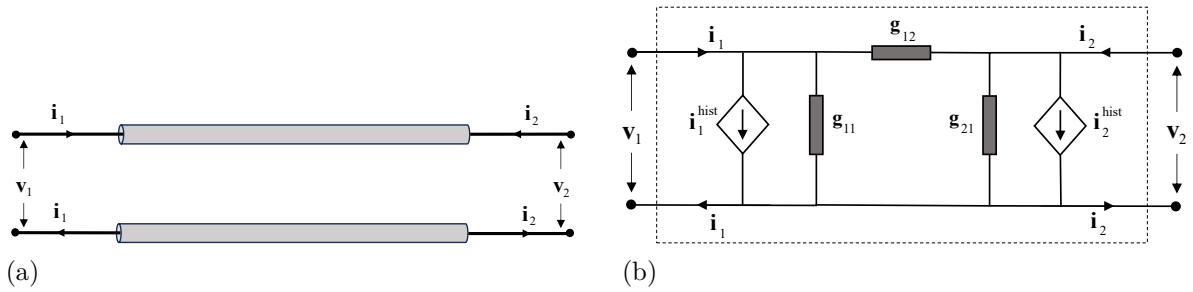


FIGURE 5.1 (a) General view of an FDNE model with 2 ports; (b) The Norton equivalent of the mentioned model

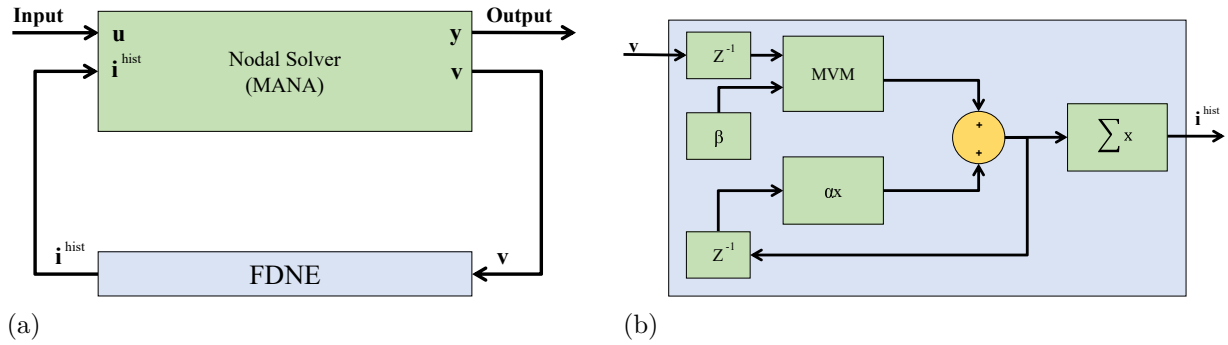


FIGURE 5.2 (a) Nodal Solver connected to FDNE; (b) High-level inside view of FDNE module

5.3.2 Hardware Description Language

RTL design is a crucial step in hardware implementation, describing digital circuit behavior using HDLs like Verilog or VHDL. However, it can be challenging and error-prone, requiring deep hardware architecture understanding. High-Level Synthesis (HLS) automates this process, taking high-level programming languages as input and generating optimized RTL code [24]. HLS reduces design time, increases productivity, and is widely used for Hardware-in-the-Loop (HIL) [29, 34]. In this paper, we leverage the advantages offered by HLS to implement our proposed FDNE model.

It is important to identify the computational demanding components and provide the appropriate algorithms and optimization techniques. In our HLS implementation, we have identified loops, and particularly nested loops, where a high level of parallelism can be exploited with the right pragmas. Therefore, we utilize the following pragmas available in the Xilinx HLS tool :

— **Pipelining :** This pragma enables a loop to be partitioned into stages, allowing for parallel execution of the loop iterations.

— **Array partitioning :** This pragma enables arrays to be partitioned into smaller sub-arrays, which can be processed in parallel.

— **Loop unrolling :** This pragma unrolls a loop, replicating the loop body multiple times and reducing the overhead of loop control operations.

5.3.3 Number Format

FPGAs provide two options for arithmetic computations : fixed-point (FXP) or floating-point (FP) formats. Both formats have their own advantages and disadvantages. Low latency data paths and fewer hardware resources are known characteristics of FXP data types. However, it has a limited dynamic range compared to the FP data format. In contrast, the FP data format offers higher accuracy and a larger dynamic range, but it requires more hardware resources and often involves more pipeline stages. The numerical resolutions employed in most related works are based on the FP format. However, in this paper, we also utilize the FXP format in order to decrease computational time and resource utilization. We provide a comprehensive evaluation of using different data types in terms of time constraints, resource utilization, and error.

5.4 Experimental Results

5.4.1 Design Space Exploration

This section describes the simulation results obtained from the implementation of the proposed method using *Vivado HLS 2019.1. Virtex UltraScale+ (xcvu13p-fhga2104-3-e)* is selected as the target device. As the first attempt, we are going to evaluate different data types, including double (DFP) and single precision floating-point (SFP), and FXP data formats. In this regard, an FDNE model with 2 ports and 16 poles, is selected for this evaluation. Among several possibilities for the FXP data type, we choose FXP 9.18 which has 27 bit-width with 18-bit fraction part. This number format can be appropriately mapped into the DSP48E blocks of Xilinx high-end FPGAs, which supply a 27x18 multiplier.

One of the most important metrics reported by the HLS tool is the *Initiation Interval (II)*. It determines the number of clock cycles before the function can initiate a new set of input reads and start to process the next set of input data. Thus, the latency is given by :

$$\text{Latency} = \text{Initiation Intervals} \times \text{Clock period} \tag{5.9}$$

It should be noted that the HLS module is simulated after the synthesis and place-and-route processes to get the clock frequency and time-step estimates close to the real ones. Hence, the practical clock frequency is the highest clock frequency that produces the same output vectors as the HLS simulation.

Furthermore, the 2-norm relative error ($\mathrm{RE}_{\text{2-norm}}$) is employed to evaluate the accuracy. Eq. 7.18 shows how this metric is computed.

$$\mathrm{RE}_{\text{2-norm}} = \frac{\|\mathbf{y}_{\text{ref}} - \mathbf{y}_{\text{eq}}\|_2}{\|\mathbf{y}_{\text{ref}}\|_2} \tag{5.10}$$

where $\mathbf{y}_{\text{ref}}$ represents the reference output and $\mathbf{y}_{\text{eq}}$ represents the output of the equivalent model. The symbol $\|\mathbf{y}\|_2$ represents the 2-norm of the vector $\mathbf{y}$.

Table 5.1 reports the timing, resource usage, and error evaluation for different data-types, including double precision floating-point (DFP), single precision floating-point (SFP), and fixed-point data format (FXP 9.18). The table shows that the latencies for DFP and SFP are 255 ns and 245 ns, respectively, while for FXP 9.18 it is 80 ns.

TABLE 5.1 Evaluate different data types in terms of time constraint, resource, and error for a 2-port, 16-pole FDNE

	Time Constraint			**Resource Utilization**				**Error Evaluation**
Data Type	Practical Clock Period (ns)	Initiation Interval	Latency (ns)	BRAM (5,376)	DSP (12,288)	FF (3,456,000)	LUT (1,728,000)	Relative Error norm-2 (%)
FXP 9.18	5	16	80	0 (0.0%)	240 (1.95%)	24,575 (0.71%)	12,981 (0.75%)	5.18×10^{-4}
SFP	5	47	235	0 (0.0%)	1,736 (14.13%)	148,552 (4.30%)	105,108 (6.08%)	1.22×10^{-5}
DFP	5	49	245	0 (0.0%)	3,976 (30.83%)	293,600 (8.04%)	251,145 (13.61%)	1.44×10^{-9}

From Table 5.1, one can confirm that sub-microsecond simulation time-steps can be achieved using all the investigated data types. Moreover, the latency is halved by switching the data type from DFP to FXP. The next columns report resource utilization and display how much BRAM, DSP, FF, and LUT are used for each data type. Using floating-point data types ends up consuming significantly more resources compared to FXP data type. The most limited resource is DSP : 30.83% in DFP, 14.13% in SFP, while it is less than 5% for FXP 9.18. Obviously, the main drawback of FXP data formats is that they have lower dynamic ranges and thus less accuracy than floating-point data formats, as shown in the reported 2-norm relative errors. However, the reported error for FXP 9.18 is still very acceptable.

Table 5.2 compares the resource utilization and time constraints of the proposed method, compared to [58], and [60], that are two related works discussed in section 5.1. The results for the proposed method have been provided in two different data types with various number of ports and poles. As shown in this table, the latency in [60] is reported in the range of microseconds. In contrast, [58] and the proposed method are sub-microseconds. For instance, the latency for a model with 1 port and 32 poles and SFP data type, has been reported $3\mu s$ in [60], and 230 ns in the proposed method. On the other hand, in [58], the reported latency for a model with 3 ports and 7 poles is 200 ns. Despite its small latency, the proposed method with different numbers of ports and poles has a latency even smaller than 200 ns for the FXP data type. Considering resource utilization, the proposed method seems advantageous too, particularly in the FXP data type, in comparison with [58] and [60].

TABLE 5.2 Comparison of the proposed method and state-of-the-art methods in terms of resource utilization and time constraint

				Resource Utilization				**Time Constraint**	
Reference	**Data Type**	**FPGA**	**(# Ports, # Poles)**	**BRAM**	**DSP**	**FF**	**LUT**	**Clock Period (ns)**	**Latency**
[58]	NSFP*	Kintex 7	(3, 7)	18 (4.0%)	198 (23.6%)	34,148 (8.4%)	31,042 (15.2 %)	4	200 ns
[60]	SFP	Virtex 7	(1, 32)	-	-	-	-	10	3 μs
	SFP		(1, 64)	0 (0.0%)	32 (1.0%)	6,000 (1.0%)	9,000 (3.0%)	10	4 μs
	SFP		(2, 32)	-	-	-	-	10	4 μs
	SFP		(2, 64)	-	-	-	-	10	4 μs
	SFP		(3, 128)	16 (1.0%)	128 (4.0%)	200,000 (32.0%)	115,000 (38.0%)	10	-
Proposed	FXP 9.18	Virtex UltraScale+	(1, 32)	0 (0.0%)	253 (2.06%)	19,733 (0.57%)	7,245 (0.42%)	5	45 ns
	SFP		(1, 32)	0 (0.0%)	1,135 (9.23%)	99,716 (2.89%)	70,881 (4.10%)	5	230 ns
	FXP 9.18		(2, 16)	0 (0.0%)	240 (1.95%)	24,575 (0.71%)	12,981 (0.75%)	5	80 ns
	SFP		(2, 16)	0 (0.0%)	1,736 (14.13%)	148,552 (4.30%)	105,108 (6.08%)	5	235 ns
	FXP 9.18		(2, 32)	0 (0.0%)	384 (3.13%)	40,361 (1.17%)	19,101 (1.11%)	5	80 ns
	SFP		(2, 32)	0 (0.0%)	2,424 (19.73%)	214,673 (6.21%)	154,740 (8.95%)	5	255 ns
	FXP 9.18		(3, 16)	0 (0.0%)	328 (2.67%)	35,016 (1.01%)	20,347 (1.18%)	5	95 ns
	SFP		(3, 16)	0 (0.0%)	2,764 (22.49%)	235,200 (6.81%)	167,201 (9.68%)	5	260 ns
	FXP 9.18		(3, 32)	0 (0.0%)	536 (4.36%)	57,470 (1.66%)	30,193 (1.75%)	5	95 ns
	SFP		(3, 32)	0 (0.0%)	5,068 (41.24%)	433,073 (12.53%)	309,008 (17.88%)	5	280 ns

*NSFP : Non-standard floating-point (34-bit mantissa, 8-bit exponent)

5.4.2 Test Case

To evaluate the behavior of the proposed model in a real text case, we consider a proposed FDNE model with two ports and 10 m length in presence of a step function, as shown in Fig. 5.3. This figure illustrates that a step function is applied to the system and the voltage is measured at the output port of the FDNE. The time-step for this simulation is considered to be 500 ns.

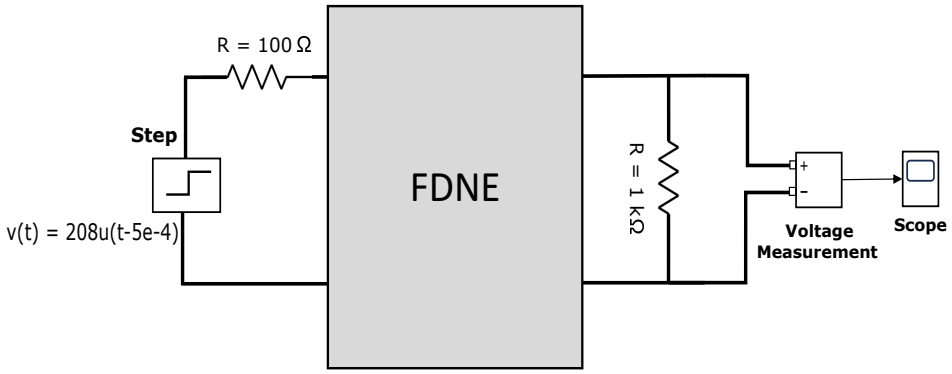


FIGURE 5.3 Test the proposed FDNE model with 2 ports and 10m length, applying a step function and measuring the output voltage at both ports

Fig. 5.4 superimposes the current for the second port. It illustrates the close-up view of the changing voltage. As can be seen, the HLS implementation matches the reference perfectly during transient.

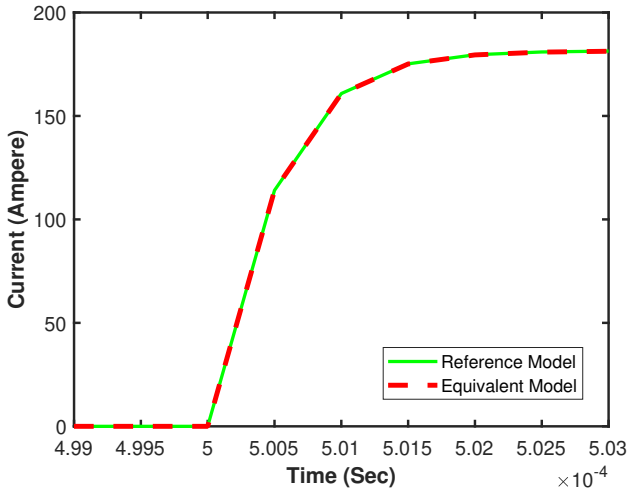


FIGURE 5.4 Close-up view of transient behavior on Port 2 when a step function is applied

5.5 Conclusion

This paper contributes to advancing real-time simulations in the aerospace industry and opens avenues for further developments in FPGA-based HIL simulations. By leveraging FDNE models and FPGAs, the proposed method offers more accurate real-time simulations of power system cable harnesses in modern aircraft, which is crucial for achieving high accuracy simulations. The FPGA-based implementation further enhances the time-step and overall performance, leveraging the parallelization capabilities of these devices. Results obtained on cables of short lengths highlight the suitability of the proposed method to offer a more accurate representation of power systems. Future work will focus on integrating the proposed models into realistic real-time power systems of aircraft, allowing for comprehensive and accurate simulations in practical operational environments.

CHAPITRE 6 ARTICLE 2 : CUFP : AN HLS LIBRARY FOR CUSTOMIZED FLOATING-POINT OPERATORS

Fahimeh Hajizadeh, Tarek Ould-Bachir, and Jean-Pierre David

Publié dans : Electronics (numéro spécial : FPGA-Based Reconfigurable Embedded Systems)

Date de parution : 18 juillet 2024

Abstract

High-Level Synthesis (HLS) tools have revolutionized FPGA application development by providing a more efficient and streamlined approach, significantly impacting digital design methodologies. Despite the capability of FPGAs to customize numerical representations in data paths, most HLS projects have focused on fixed-point precision, while floating-point representations remain limited to vendor-provided single, double, and half-precision formats. This paper proposes a customized floating-point library compatible with HLS to address these limitations. This library allows programmers to define the number of exponent and mantissa bits at compile time, providing greater flexibility and enabling the use of mixed precision. Moreover, this library includes optimized implementations of common components such as vector summation (VSUM), dot-product (DP), and matrix-vector multiplication (MVM). Results demonstrate that the proposed library reduces latency and resource utilization compared to vendor IP blocks, particularly in VSUM, DP, and MVM operations. For example, the mvm operation involving a 32×32 matrix, using vendor IP requires 22 clock cycles, whereas CuFP completes the same task in just 7 clock cycles, using approximately 60% fewer DSPs, 10% fewer LUTs, and 60% fewer FFs.

Keywords

floating-point ; high-level synthesis (HLS) ; FPGA ; custom precision ; customized floating-point ; custom operation ; vector summation (VSUM) ; dot-product (DP) ; matrix-vector multiplication (MVM)

6.1 Introduction

The High-Level Synthesis (HLS) approach is pivotal in making FPGA programming more accessible and significantly enhancing design productivity. HLS tools empower developers to

describe digital system behaviors through high-level programming languages like C or C++, freeing them from low-level hardware details. This abstraction allows a focus on system-level functionality, leading to faster design iterations. Moreover, HLS tools automate various optimization processes, reducing manual effort and accelerating development cycles [67].

The latest HLS tools have built-in support for single- and double-precision floating-point types and operations [68]. There are various sources of floating-point IP available for FPGA designs. One option is to use IP core generators provided by vendors [69], such as the floating-point libraries from Xilinx [69, 70] and Intel [71]. However, floating-point cores are encapsulated as "black-box" entities in HLS tools, and developers encounter limitations in accessing and modifying their internal configurations, which restricts their ability to fine-tune precision for specific applications [72–74]. As will be seen hereafter, enabling custom precision for floating-point operations at a high level is one of the main contributions of our proposed library. Despite advancements in intra-cycle scheduling of combinational components, which deals with the scheduling of unified pipelined blocks in HLS, internal pipeline registers within IP blocks may still impact block-level scheduling and pose efficiency challenges, particularly in coordinating diverse operations within the pipeline to maximize throughput while meeting timing constraints [47]. A more flexible approach will be possible with our proposed library, as the internal architecture of our operators, including the pipeline registers, will be editable by the user.

In the register transfer level (RTL) design, it is relatively straightforward to specify custom-precision floating-point operations, and optimizing floating-point types can effectively reduce resource usage without compromising accuracy [1, 25, 75, 76]. One of the works at the RTL level is proposed in [77], where floating-point operator entities receive general inputs that define floating-point types, and the data-path widths are established during the elaboration phase. Flopoco [42] stands out as a significant research effort at the RTL level and serves as a foundation for numerous other researches [46, 78–80]. It focuses on efficient floating-point arithmetic implementations. In contrast, FPGA customization for floating-point operations is less explored in many HLS methodologies, especially compared to fixed-point representations. This is evident because HLS tools typically support a limited set of vendor-provided floating-point types, such as half-, single-, and double-precision. Some works focus on high-level, customized floating-point representations [46, 48] but offer little control over certain hardware optimizations, such as latency or resource utilization.

These observations highlight the need for greater customization and flexibility in floating-point operations within HLS techniques to achieve optimal scheduling, efficient resource utilization and lower latencies. In that regard, minimizing latency can be a crucial target for

such applications as hardware-in-the-loop (HIL) [81]. Many applications require extensive floating-point computations but suffer from high latency and excessive resource utilization without the full precision of standard floating-point formats [3, 82, 83]. Customized floating-point implementations, by tailoring precision and format, were shown to be a means to improve latency [84].

This paper introduces CuFP, an HLS-based library for customized floating-point operators. A key attribute of this HLS-based library is its flexibility, allowing users to customize floating-point types for their calculations, providing users with enhanced control over precision. Another significant contribution is the provision of dedicated operators, such as vector sum, dot-product, and matrix-vector multiplication operations, which are crucial for numerous applications. This integration sets this library apart from others. The proposed library significantly reduces latency and resource utilization for the aforementioned operators. CuFP is fully compatible with Xilinx tools via a command-line interface, enabling users to specify parameters and generate corresponding IP cores seamlessly, enhancing usability and facilitating efficient integration into FPGA designs. Incorporating CuFP into other projects is straightforward, as users can simply include a header file, ensuring effortless integration and user-friendly implementation. Its platform independence and open-source nature promote transparency and ease of debugging while maintaining performance levels (`https://github.com/FahimeHajizadeh/Custom-Float-HLS`, accessed on 16 July 2024).

The remainder of this paper is organized as follows : Section 6.2 presents an overview of floating-point numbers and explains some recent works related to customized floating-point numbers. Section 6.3 describes the development flow in detail. This section explains a detailed implementation of the proposed method and dedicated operations. Section 6.4 presents the results for primary operations and dedicated operations compared to others. A discussion and conclusion are given in Sections 6.5 and 6.6, respectively.

6.2 Background

6.2.1 Floating-Point Format

The IEEE-754 floating-point standard is widely used and recognized for representing floating-point numbers [37]. It comprises three main components : the sign bit, the exponent, and the mantissa. The value of a floating-point number is calculated using the formula :

$$\text{Value} = (-1)^{\text{s}} \times 2^{\text{e}} \times \text{m} \tag{6.1}$$

where s indicates the sign of the number, with 0 representing positive and 1 representing negative, e is the exponent, typically stored in a biased form, which scales the number by a power of two, allowing for a wide range of values, and m is the normalized mantissa $m \in [1, 2)$. The mantissa represents the significant digits of the number, providing the necessary precision. Figure 6.1 illustrates the binary floating-point representation, where a number is given on Wfp bits. The sign bit occupies one bit, the biased exponent occupies We bits, and the fraction occupies Wf bits, making $\text{Wfp} = 1 + \text{We} + \text{Wf}$.

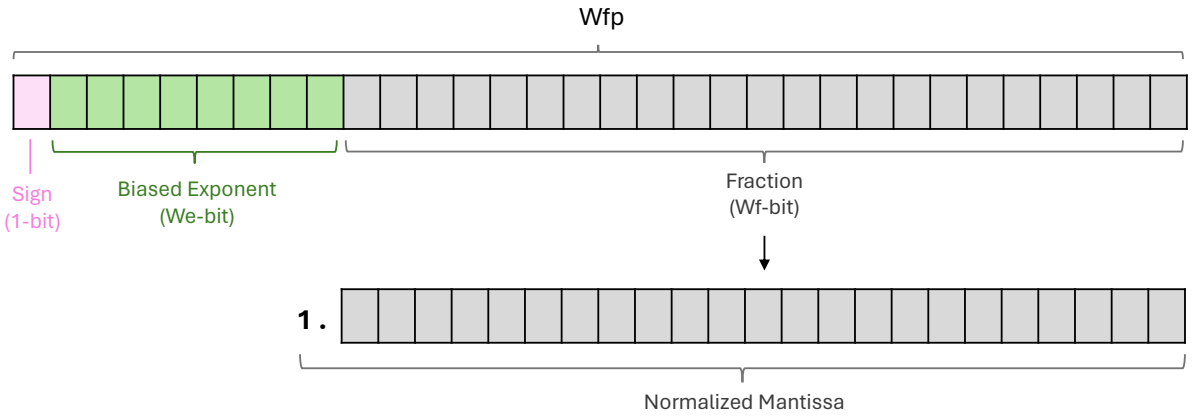


FIGURE 6.1 Structure of the IEEE-754 floating-point standard.

The IEEE-754 standard defines various precision levels, including half-precision (16 bits), single-precision (32 bits), double-precision (64 bits), and quadruple-precision (128 bits), each with distinct bit widths, as shown in Table 6.1.

TABLE 6.1 Specifications of IEEE 754 Floating-Point Formats

Format	**Word Size (bits)** ($\mathbf{W_{fp}}$)	**Exponent Bits** ($\mathbf{W_e}$)	**Fraction Bits** ($\mathbf{W_f}$)	**Bias**
Half-Precision	16	5	10	15
Single-Precision	32	8	23	127
Double-Precision	64	11	52	1023
Quadruple-Precision	128	15	112	16,383

Allocating more bits to the exponent increases the dynamic range of representable numbers, allowing the number format to handle a wider range of magnitudes, accommodating both

very large and very small numbers. Conversely, allocating more bits to the mantissa enhances precision, as more bits in the fraction allow for finer granularity of representable values.

6.2.2 Custom Formats

While standard formats are widely used, there are situations where bit-width customization is beneficial [39, 40]. Custom floating-point formats can achieve a trade-off between dynamic range, speed, area, and numerical resolution, depending on the application's specific requirements. Consequently, customized floating-point garners considerable attention.

RTL Libraries

VFLOAT [41], known as the Variable Precision Floating Point library, offers a comprehensive suite of variable-precision floating-point cores designed to perform fundamental arithmetic operations such as addition, subtraction, multiplication, and division. Additionally, it integrates conversion operators to facilitate transitions between floating-point and fixed-point representations. While VFLOAT serves as a valuable reference library for testing and validating floating-point cores, it is intended primarily for validation purposes rather than for use in actual designs.

FloPoCo [42, 43], short for "FLoating Point Operator COres," is an open-source framework dedicated to producing optimized arithmetic operators specifically tailored for FPGA setups. This platform automates the generation of VHDL or Verilog code for a wide range of arithmetic functions, including addition, subtraction, and multiplication. FloPoCo creates non-encrypted VHDL cores with templates designed for integer, fixed-point, floating-point, and complex types.

HLS Libraries

The integration of custom floating-point implementations into HLS methodologies has been limited despite their long-standing use in FPGAs, the focus of many HLS tools on fixed-point arithmetic, and the challenge of achieving optimal performance and resource utilization. Several methods exist to address the need for soft floating-point libraries within HLS [44]. Still, these libraries cannot often generate IP cores at compile-time, restricting them to predefined operator widths. One approach explores using HLS to develop non-standard floating-point operations, such as summation [45], but it does not support heterogeneous precision.

Template HLS (THLS) [46] addresses customized floating-point operations at a high level using C++ templates. THLS allows compile-time selection of exponent and fraction widths

and supports mixed precisions for input arguments and result types [47]. While it offers a unique implementation for both simulation and synthesis, it primarily focuses on basic operations, leaving room for further optimization and additional useful operations.

TrueFloat [48], integrated with the open-source HLS tool Bambu [49], represents another significant advancement. This integration facilitates new optimization opportunities and generates equivalent representations at a higher level of abstraction. TrueFloat simplifies the translation between different floating-point encodings and offers a straightforward process through simple command-line options.

Another HLS library is the Fused Vector FP architecture [50]. In this study, the authors develop a parameterized fused many-term floating-point dot product architecture optimized for high-level synthesis. This strategy combines the efficiency of a fused dot-product structure with the adaptability of high-level synthesis, allowing precise tuning for optimal performance and resource usage across different formats and hardware configurations. While this architecture showcases significant advancements, it also has notable limitations. One primary drawback is its lack of support for heterogeneous formats, which limits its flexibility and application in diverse computational tasks. Moreover, it should be noted that unlike previous works that are FPGA-based, Fused Vector FP is designed for ASIC implementations.

6.3 Implementation Methodology

6.3.1 Development Flow

Developing customizable floating-point operations requires efficient, optimized C++ code adhering to HLS guidelines. Using directives and pragma annotations enhances hardware performance, area, and power consumption. Clear specifications of input/output formats, precision requirements, complex operations, and performance constraints are essential for the design and implementation phases. Once defined, the core logic is implemented in C++.

Figure 6.2 illustrates the HLS workflow of the CuFP framework, demonstrating user interaction to create desired IP. After compiling and debugging the C-based code, the HLS tool packages the function into hardware IP for RTL-based projects. CuFP supports heterogeneous number formats, allowing arbitrary bit-widths for inputs and outputs via a template-based interface, ensuring hardware implementation aligns with application needs and maximizing efficiency and precision.

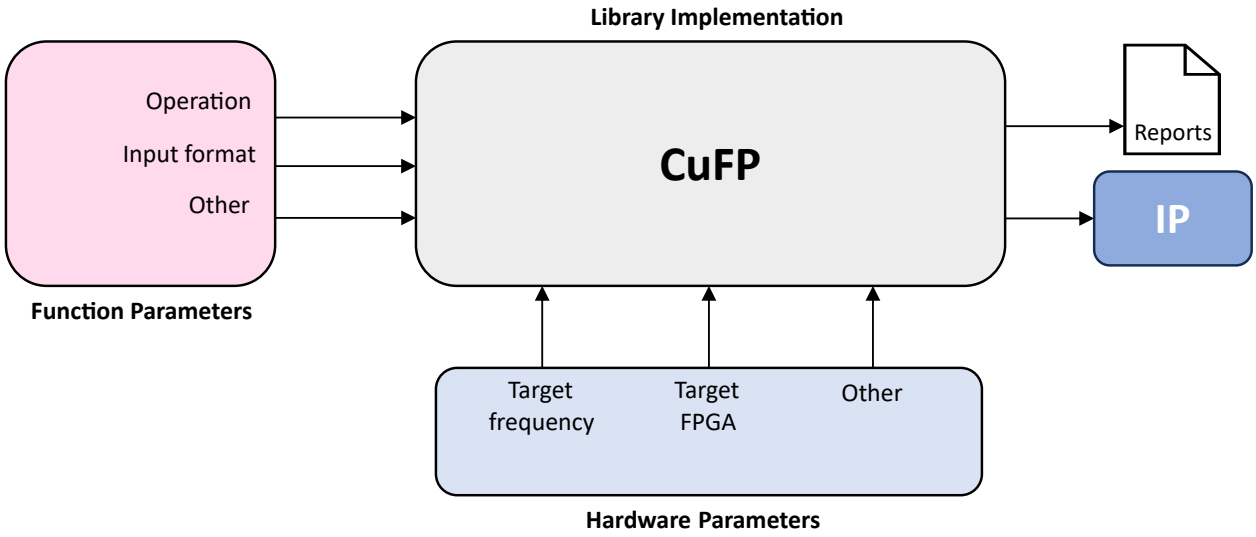


FIGURE 6.2 HLS workflow of the CuFP framework.

Customizable bit-width formats benefit applications with varying precision and resource constraints. High-precision computations may require extended bit-widths to minimize errors, while real-time tasks may prioritize lower latency and reduced resource usage with narrower bit-widths. Our approach leverages the HLS tool's built-in libraries, ensuring lightweight, independent implementations, simplifying development, and enhancing project compatibility.

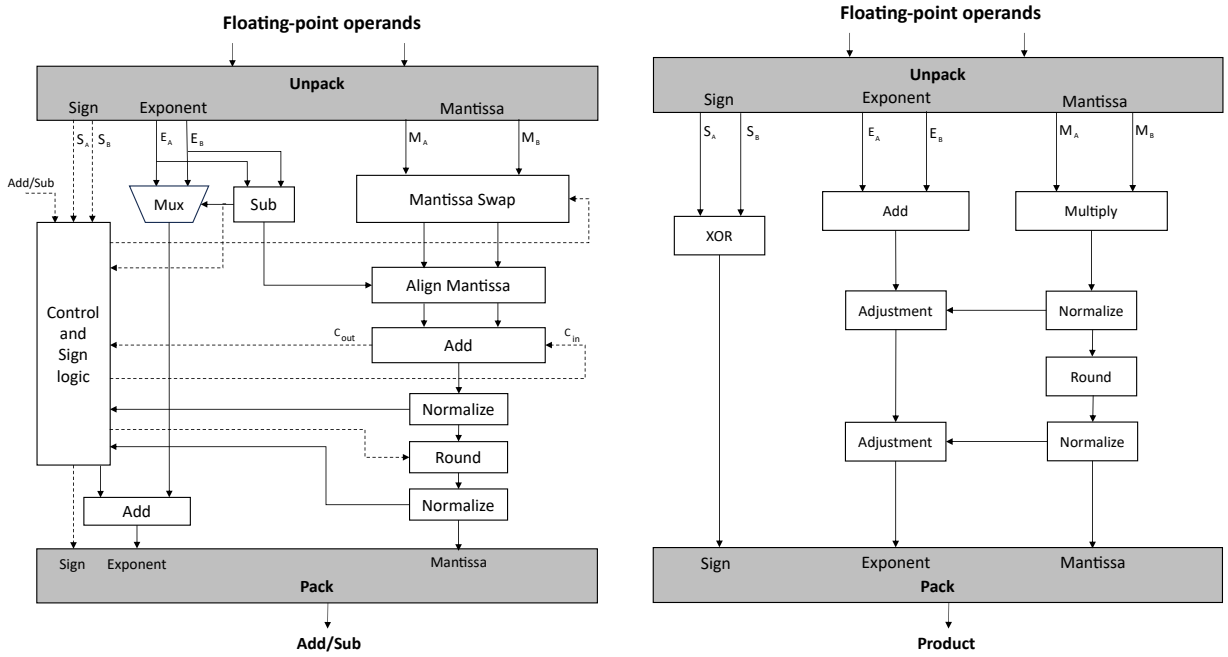
A key achievement of the library is providing dedicated volumetric vector and matrix operations, specifically vector summation (VSUM), dot-product (DP), and matrix-vector multiplication (MVM), utilizing HLS built-in libraries. These operations are crucial for high-performance computing applications like scientific simulations, 3D graphics processing, and machine learning algorithms. Customizing bit-widths allows users to achieve desired precision, manage resource usage, and ensure efficient computation for various operations on large vectors.

The CuFP project can be utilized at both a low and high level, depending on the requirements. The proposed library can be imported into an HLS project for high-level use or exported as an RTL IP and instantiated into an RTL design. We created a script that exports an RTL IP from the source code and synthesizes and implements it automatically. This approach can serve as inspiration for developing an FPGA-based project using the CuFP library.

6.3.2 Standard Floating-Point Operations

Unlike fixed-point numbers, floating-point numbers offer an extensive dynamic range by allowing the radix point to float. However, they must be aligned for almost every arithmetic operation, requiring unpacking before computation and packing afterward. Additionally, values must be normalized after each numerical computation when stored in memory or registers.

Figure 6.3 illustrates a block-level schematic of addition and multiplication for floating-point numbers in standard formats. For addition, the process begins by aligning the exponents of the two values, which involves shifting the smaller exponent's mantissa and updating the exponent. Once aligned, the mantissas are added or subtracted. In multiplication, the mantissas are multiplied, and the exponents are added to determine the preliminary product and exponent.



(a) (b)

FIGURE 6.3 (a) Standard floating-point addition; (b) Standard floating-point multiplication [85]

CuFP follows these steps with modifications to enable additional capabilities. Efficient HLS development requires knowledge of the target FPGA's resources and the HLS tool's mapping strategies. While algorithms in high-level languages may execute efficiently on x86 CPUs, HLS tools may not generate efficient FPGA logic. FPGAs benefit from parallel architectures, so dividing algorithms into parallelizable portions and scheduling them accordingly can enhance

performance. These optimizations are typically performed by the HLS tool, but developers must provide useful hints to guide the compiler.

6.3.3 CuFP : Detailed Implementation

The CuFP library, designed with a templated C++ format, supports a wide range of custom floating-point data types. This approach offers several advantages, including determining the optimal width for operands in each operation and using heterogeneous floating-point data types with different exponent and mantissa sizes. This flexibility is beneficial for developing algorithms focusing on efficiency or low latency.

The performance and efficiency of the final RTL generated by the HLS tool depend heavily on the coding style. While an object-oriented approach in C++ may introduce some overhead compared to a procedural C-style format, it results in cleaner, more maintainable, and extensible code. Balancing these approaches can enhance both performance and code quality. In CuFP, we utilized a mixture of both approaches to keep the latency and resources as low as possible while keeping the source code easy to use, modify, and extend. In this direction, we avoided using extra classes for different data types and instead used a templated class with various member functions that implement primary floating-point operations like addition, subtraction, and multiplication, allowing us to extend the library to support more complex operations such as dot-product, vector summation, and matrix-vector multiplication.

The CustomFloat class template in the CuFP namespace shown in Listing 6.1 defines a custom floating-point number with customizable bit-width (`WL`) and mantissa size (`MS`). It includes member variables for the mantissa (`mnts`), exponent (`exp`), and sign (`sign`). The class provides several constructors and utility functions to allow users to convert different data types together and work with them easily. We deliberately omitted implementation details to concentrate on the most important code sections in this code sample and the subsequent code snippets throughout the paper.

Listing 6.1 The body of CustomFloat template class.

```
1  namespace CuFP {
2  template <int WL, int MS>
3  class CustomFloat {
4  public:
5      ap_uint<MS+1> mnts; // Mantissa
6      ap_uint<WL-MS> exp; // Exponent
7      bool sign;          // Sign bit
8      CustomFloat()  { ... } // Default constructor
9      // Constructor by given values
10     CustomFloat(ap_uint<MS> m, ap_uint<WL-MS> e, bool s)  { ... }
11     // Constructor with double input
12     CustomFloat(double val)  { ... }
13     // Copy Constructor
14     CustomFloat(const CustomFloat<WL, MS>& copy) { ... }
15     // Getter function to return the value in double
16     double getDouble() const { ... }
17     ...
18 };
19 }
```

Primary Operations

The templated function `mul` performs multiplication on two custom floating-point numbers stored in the CustomFloat class. A pseudo-code of this function is given in Listing 6.2. It starts by multiplying the mantissas of the operands `x` and `y`, resulting in a potentially larger intermediate mantissa (line 5). The function then normalizes and rounds the resulting mantissa. It calculates the number of bits to shift based on the mantissa sizes (line 8) and performs a right shift, either rounding or truncating the mantissa based on whether `EN_ROUNDING` is defined (lines 9–13). Next, the function adjusts the exponent by adding the exponents of `x` and `y` along with any overflow bits detected from the normalization process (lines 16–17). It determines the sign of the result by XORing the signs of the operands (line 20). Finally, the function constructs and returns a new `CustomFloat`, completing the multiplication operation (line 22).

Listing 6.2 A pseudo-code of CustomFloat multiplication operation.

```
1  template<int WR, int MR, int WX, int MX, int WY, int MY>
2  CustomFloat<WR, MR> mul(CustomFloat<WX, MX> x,
3                          CustomFloat<WY, MY> y) {
4      // Multiply the mantissas
```

```
5       mnts = multiply(x.mnts, y.mnts);
6
7       // Normalizing and rounding
8       shift_mnts = MX + MY - MR;
9   #ifdef EN_ROUNDING
10      shift_and_round(mnts, shift);
11  #else
12      shift_and_truncate(mnts, shift);
13  #endif
14
15      // Exponent adjustment
16      ov_bits = extract_overflow_bits(mnts);
17      exp = add(x.exp, y.exp, ov_bits);
18
19      // Check sign bit
20      sign = xor(x.sign, y.sign);
21
22      return CustomFloat(mnts, exp, sign);
23  }
```

The templated function `sum` performs addition on two custom floating-point numbers stored in the `CustomFloat` class. A pseudo-code of this function is given in Listing 6.3. It begins by comparing the exponents of the operands and swapping them if necessary, setting the result's exponent to this larger value (lines 5–7). It calculates the difference between the exponents to align the mantissas, shifting the smaller exponent's mantissa to the right, either rounding or truncating it based on whether `EN_ROUNDING` is defined (lines 10–15). Then, it extends the mantissas to the same bit-width to ensure they can be added correctly (line 18). After applying the signs to the mantissas, it sums them up and determines the sign of the resulting mantissa (lines 21–24). The function then normalizes the result ; if no overflow bits are present, it finds the position of the leading one, left-shifts the mantissa to normalize it, and adjusts the exponent accordingly. If overflow bits exist, it right-shifts the mantissa by one bit and increments the exponent, again considering rounding if defined (lines 27–39). Finally, the function constructs and returns a new `CustomFloat` object, completing the addition operation (line 40).

Listing 6.3 A pseudo-code of CustomFloat summation operation.

```
1   template<int WR, int MR, int WX, int MX, int WY, int MY>
2   CustomFloat<WR, MR> sum(CustomFloat<WX, MX> x,
3                           CustomFloat<WY, MY> y) {
4       // Swap operands based on the greater exponent
```

```
5       if (compare(x.exp, y.exp))
6           swap(x, y);
7       exp = x.exp;
8
9       // Align mantissas
10      diff = sub(x.exp, y.exp);
11  #ifdef EN_ROUNDING
12      rshift_and_round(mnts, diff);
13  #else
14      rshift_and_truncate(mnts, diff);
15  #endif
16
17      // Extend the shorter mantissa
18      extend(x.mnts, y.mnts, MX, MY);
19
20      // Sum the two mantissas
21      apply_sign(x.mnts, x.sign);
22      apply_sign(y.mnts, y.sign);
23      mnts = add(x.mnts, y.mnts);
24      sign = check_sign(mnts);
25
26      // Normalizing and rounding
27      ov_bits = extract_overflow_bits(mnts);
28      if (ov_bits == 0) {
29          lod = lead_one_pos(mnts);
30          lshift(mnnts, lod);
31          sub(exp, lod);
32      } else if (ov_bits > 1) {
33  #ifdef EN_ROUNDING
34          rshift_and_round(mnts, 1);
35  #else
36          rshift_and_truncate(mnts, 1);
37  #endif
38          exp = add(exp, 1);
39      }
40      return CustomFloat(mnts, exp, sign);
41  }
```

The CuFP library uses flattened code or template-based recursive functions instead of loops in sub-modules and sub-functions. While this approach may demand more resources, it results in highly competitive performance. This makes CuFP ideal for applications such as real-time simulation, signal processing, and autonomous systems that require fast, adaptable floating-

point operations with reasonable resource usage.

6.3.4 Customized Vector Operations

In computation, scientific, and engineering fields, operations like matrix-vector multiplication, dot-product, and vector summation are crucial for performance. These operations underpin various algorithms and numerical computations, from simple arithmetic to complex tasks in statistical analysis, signal processing, image processing, and machine learning. Efficient execution of these operations enhances application performance, enabling faster computations and better resource utilization. We have incorporated these operations as custom blocks within the CuFP library to optimize latency and resource use, leveraging customized floating-point arithmetic.

Vector Summation

Vector summation is a fundamental operation in computational mathematics and data processing that entails adding whole items within a vector. Consider an N-sized vector represented as $\mathbf{x} = [x_1, ..., x_{N-1}, x_N]$. The vector summation operation is defined as follows :

$$\sum_{i=1}^{N} x_i = x_1 + ... + x_{N-1} + x_N \tag{6.2}$$

The templated function `vsum` accumulates N elements of an array of custom floating-point numbers (`CustomFloat<WR, MR>`). Listing 6.4 provides a simplified representation of the vsum body function. It begins by determining the additional bits needed to safely sum the mantissas (line 5). The function then extracts the exponents from each array element, identifies the maximum exponent (`max_exp`), and assigns it as the result exponent (lines 8–10). It aligns all mantissas to this maximum exponent by computing exponent differences and shifting the mantissas accordingly (lines 13–14). After applying the signs to the mantissas, they are summed using a recursive template adder, and the resulting sign is determined (lines 17–19). The rest is quite similar to the sum operation elaborated in the previous subsection.

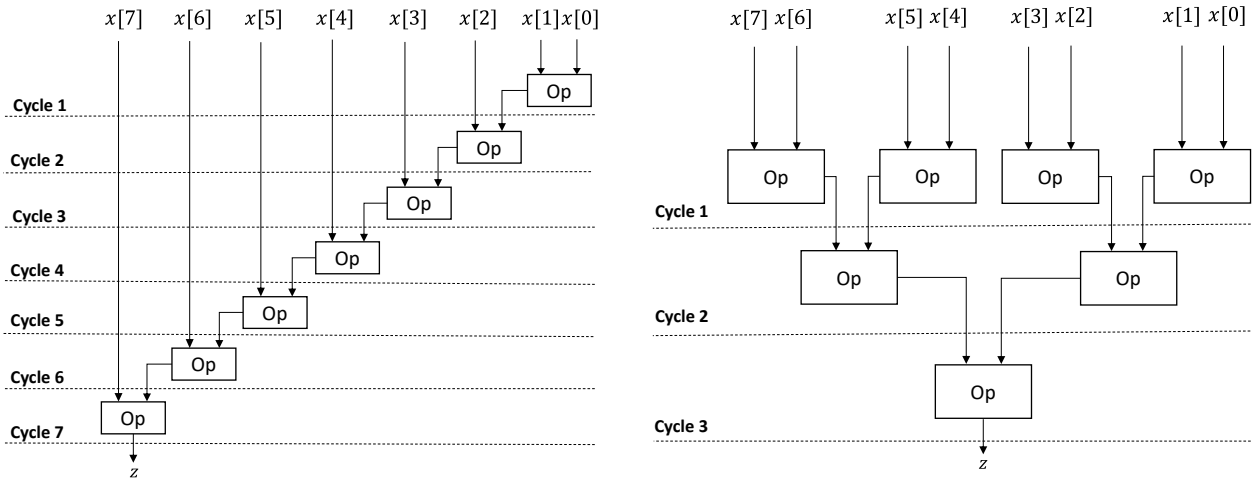
A key aspect of developing generic vector summation is its ability to accumulate a variable set of customized floating-point numbers. Given that the input array can vary in size, we needed loops or similar constructs to perform essential sub-functions such as determining the maximum exponent (line 9) or summing the mantissas (line 18). A straightforward approach is to use loops with unrolling directives. However, the Vivado HLS tool typically creates an unbalanced tree structure for array-reduction operations like an accumulator when

unrolling the corresponding loops. Figure 6.4 illustrates the difference between unbalanced and balanced tree hierarchies for executing array-reduced operations on an 8-sized array. Both consume 7 binary operators but in different cycles, resulting in the unbalanced version having a longer critical path and inefficient performance. Although there is a directive (`unsafe_math_optimizations`) [86] to balance match expressions under certain conditions, we designed a template-based recursive function structure to reduce reliance on tool features and ensure the operations are consistently optimized.

Listing 6.4 A pseudo-code of CustomFloat vector summation operation.

```
template<int WR, int MR, int N>
CustomFloat<WR, MR> vsum(CustomFloat<WR, MR> x[N])
{
    // Extract the necessary extra bits to safely add the mantissas
    exbit = select_extra_bit(N);

    // Find the maximum exponent
    v_exp[] = vector_of_exp(x);
    max_exp = find_max(v_exp);
    exp = max_exp;

    // Align all the mantissas based on the maximum exponent
    v_d_exp[] = diff_exp(v_exp, max_exp);
    v_mnts[] = shift_mantissa(x, v_d_exp);

    // Sum mantissas of CuFP numbers in the array x
    apply_sign(v_mnts, x);
    mnts<MR+exbit> = recursive_template_adder(v_mnts);
    sign = check_sign(mnts);

    // Normalizing and rounding
    ov_bits = extract_overflow_bits(mnts);
    if (ov_bits == 0) {
        lod = lead_one_pos(mnts);
        lshift(mnnts, lod);
        sub(exp, lod);
    } else if (ov_bits > 1) {
#ifdef EN_ROUNDING
        rshift_and_round(mnts, ov_bits);
#else
        rshift_and_truncate(mnts, ov_bits);
#endif
        exp = add(exp, ov_bits);
```

```
34      }
35      return CustomFloat(mnts, exp, sign);
36  }
```



(a) Unbalanced tree binary operations

(b) Balanced tree binary operations

FIGURE 6.4 Binary operation trees : (**a**) Unbalanced structure ; (**b**) Balanced structure.

Dot-Product Operation

The dot-product, also known as the scalar product or inner product, calculates the sum of the products of corresponding elements in two vectors. Mathematically, the dot product of two vectors $\mathbf{x}$ and $\mathbf{a}$ of size N is computed as the sum of the products of their corresponding elements :

$$\langle \mathbf{x}, \mathbf{a} \rangle = \mathbf{x}^T \cdot \mathbf{a} = \sum_{i=1}^{N} x_i \cdot a_i = x_1 \cdot a_1 + x_2 \cdot a_2 + ... + x_N \cdot a_N \tag{6.3}$$

As shown in Equation (6.3), dot-product involves multiplying two vectors and then summing these products. Therefore, Developing the generic dot-product operation was carried out by `mul` and `vsum` operations. Listing 6.5 demonstrates how to use these functions to complete the dot-product operation. We first multiply the array's corresponding elements using an unrolled loop. All multiplications are executed in parallel in this stage. The produced array is then passed to the `vsum` operator, which computes the final result. Additionally, the inline and pipeline directives are employed to prevent component reuse and ensure that the process is fully pipelined.

Listing 6.5 A pseudo-code of CustomFloat dot-product operation.

```
template<int WR, int MR, int N>
CustomFloat<WR, MR> dp(CustomFloat<WR, MR> a[N],
                       CustomFloat<WR, MR> x[N])
{
#pragma HLS INLINE recursive
#pragma HLS PIPELINE II=1
    CustomFloat<WR, MR> v[N];
    for (int i = 0; i < N; i ++) {
#pragma HLS UNROLL factor=N
        v[i] = mul<WR, MR>(a[i], x[i]);
    }
    return vsum<WR, MR, N>(v);
}
```

Matrix-Vector Multiplication

Matrix-vector multiplication is another fundamental operation in linear algebra that involves multiplying a matrix by a vector to produce a new vector. This operation plays a pivotal role in various mathematical and computational tasks. Conceptually, matrix-vector multiplication consists of taking each row of the matrix and performing a dot-product with the vector, resulting in the corresponding element of the resulting vector as shown in Equations (6.4) and (8.6).

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NN} \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{6.4}$$

The result of multiplying matrix $\mathbf{X}$ by vector $\mathbf{y}$ is calculated as follows :

$$\mathbf{z} = \mathbf{X}\mathbf{y} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1N} \\ x_{21} & x_{22} & \cdots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NN} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_{11}y_1 + x_{12}y_2 + \cdots + x_{1N}y_N \\ x_{21}y_1 + x_{22}y_2 + \cdots + x_{2N}y_N \\ \vdots \\ x_{N1}y_1 + x_{N2}y_2 + \cdots + x_{NN}y_N \end{bmatrix} \tag{6.5}$$

So, the result of the matrix-vector multiplication $\mathbf{X}\mathbf{y}$ is an N-element vector where each element is obtained by taking the dot-product of the i-th row of the matrix $\mathbf{X}$ with the vector $\mathbf{y}$. Listing 6.6 represents the implementation of `mvm` using dot-product operation inside an unrolling loop.

Listing 6.6 A pseudo-code of CustomFloat matrix-vector multiplication operation.

```
template<int WR, int MR, int N>
void mvm(CustomFloat<WR, MR> x[N][N],
         CustomFloat<WR, MR> y[N],
         CustomFloat<WR, MR> z[N])
{
#pragma HLS INLINE recursive
#pragma HLS PIPELINE II=1
  for (int i = 0; i < N; i++) {
#pragma HLS UNROLL factor=N
    z[i] = CuFl::dp<WR, MR, N>(&x[i][0], y);
  }
}
```

6.3.5 CuFP Automation

An automated process employing TCL and bash scripts was created to simplify the synthesis and implementation of customized floating-point operations in HLS. Users can select specific operations like `sum`, `mul`, `vsum`, `dp`, or `mvm` and set design parameters, including customized bit-widths. The automation compiles the HLS code, synthesizes the design, and exports the generated RTL as an IP block, ready for integration into larger projects. It also supports the Vivado Design Suite implementation with specified hardware parts and clock frequencies, as shown in Figure 6.2.

Encapsulating the entire process in a single bash script command allows users to quickly generate RTL designs without delving into library details, reducing errors and manual mistakes. It enables easy exploration of different configurations, facilitating the identification of the most suitable version for specific requirements. This ensures consistent quality results and accelerates the design workflow.

6.4 Experimental Results

This section presents the experimental results of implementing the proposed CuFP library with Vivado HLS 2019.1. It should be noted that our open-source library does not rely on third-party dependencies or use specific features of a particular version of Xilinx tools. This ensures compatibility and seamless operation across different versions of Xilinx tools. The target device chosen is the Virtex UltraScale+ (xcvu9p-fsga2104-3-e). Although the design space for evaluating a generic library like CuFP is extensive, we focused on assessing it in the most common and impactful areas. We aimed to provide valuable quantitative and qualitative

experimental results to help researchers select the most suitable option for their work.

The operations in the CuFP library are designed with full pipelining, allowing new input values to be fed every clock cycle for continuous data processing. However, the initiation interval can be increased by applying the appropriate directives in the code, offering flexibility in scheduling operations. This strategy may result in better resource utilization. For the sake of fairness in the comparison, the Vendor's IPs are likewise set to be fully pipelined in the following results too.

For each circuit reported in this paper, a set of registers is present at the input and output stages to ensure accurate signal capture and readiness for subsequent stages. For illustrative purposes, a circuit with three levels of the pipeline actually has two computing stages surrounded by registers.

6.4.1 Primary Operations

We evaluate the impact of fraction width variation on the performance of floating-point arithmetic operations. We consider the fundamental arithmetic operations of `sum` and `mul` in both versions of rounding (CuFP (r)) and truncation (CuFP(t)).

The 2-norm relative error ($\mathrm{RE}_{\text{2-norm}}$) is employed to evaluate the accuracy. Equation 7.18 shows how this metric is computed.

$$\mathrm{RE}_{\text{2-norm}} = \frac{\|out_{\text{ref}} - out_{\text{com}}\|_2}{\|out_{\text{ref}}\|_2} \tag{6.6}$$

where out_{ref} represents the reference output and out_{com} represents the computed output. The symbol $\|out\|_2$ represents the 2-norm of the output.

Table 8.1 presents the 2-norm relative error percentages for the CuFP library in both truncation and rounding modes, as well as for the vendor IP, compared to double-precision floating-point for 100,000 random numbers. The results indicate that for both `sum` and `mul` operations, the CuFP library in truncation mode shows slightly higher errors than the rounding mode, which matches the performance of the vendor IP. Overall, the errors are very small, demonstrating the high accuracy of the proposed library in both modes.

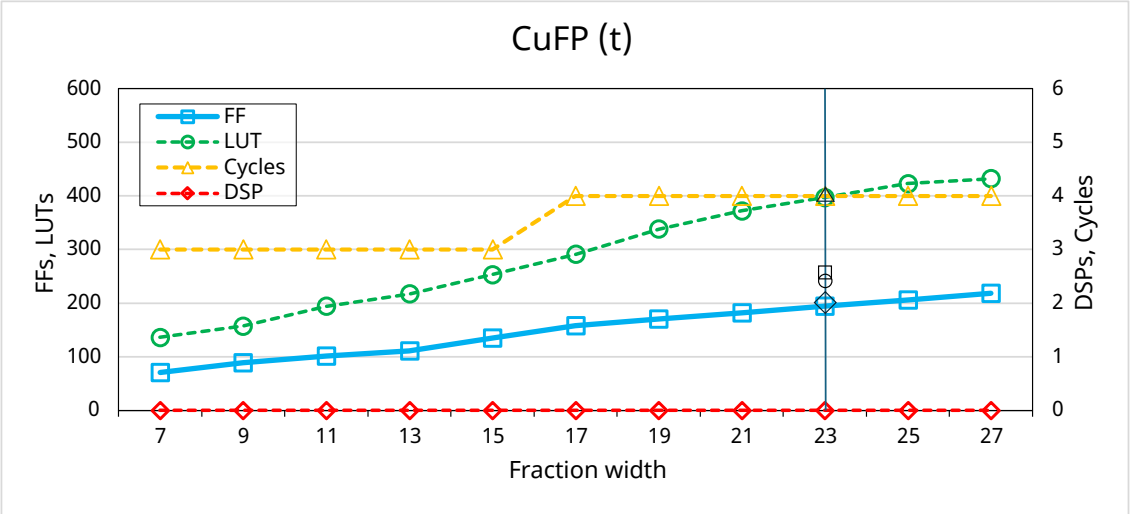
TABLE 6.2 Error evaluation of CuFP (8, 23) in truncation and rounding modes, and Vendor IP (single-precision floating-point), based on 100,000 random numbers.

Operation	**Variant**	**2-Norm Relative Error (%)**
Sum	CuFP (t)	2.68×10^{-6}
	CuFP (r)	2.02×10^{-6}
	Vendor IP	2.02×10^{-6}
Mul	CuFP (t)	6.35×10^{-6}
	CuFP (r)	6.03×10^{-6}
	Vendor IP	6.03×10^{-6}

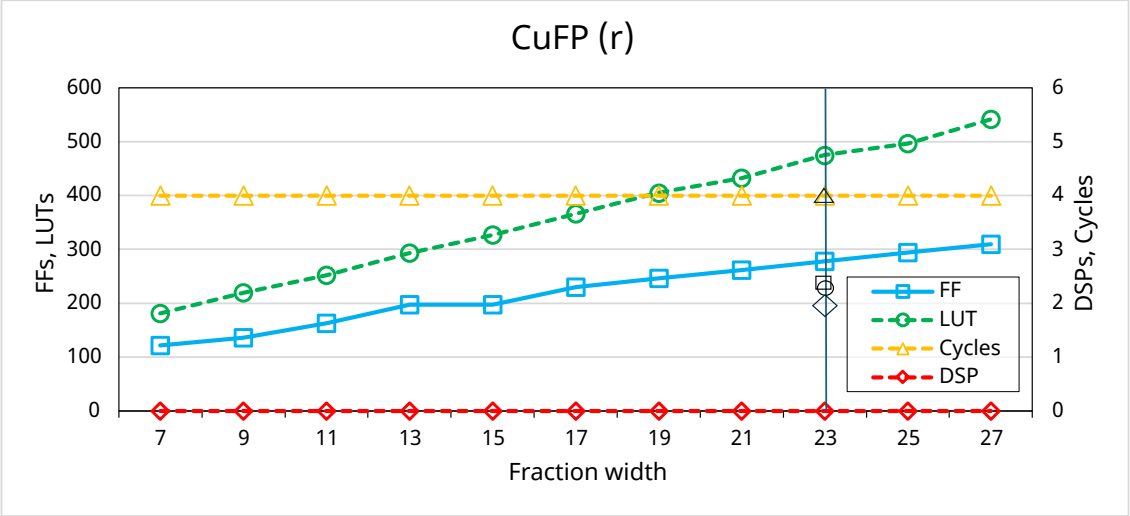
Figure 6.5 illustrates CuFP's resource utilization and cycle counts for the `sum` operation across different fraction widths at 200 MHz. Resource utilization in truncation mode is generally lower than in rounding mode due to the absence of additional rounding logic. Look-up table (LUT) utilization increases linearly in both modes, while flip-flops (FF) utilization also rises, albeit slower. CuFP's use of integer-based operators for adding fractions in the `sum` operation results in no digital signal processing (DSP) block allocation by the HLS tool. The cycle count is steady at 4 cycles in rounding mode, while truncation mode maintains 3 cycles up to 15 fraction bits, increasing by one cycle thereafter. This highlights the efficiency trade-offs between rounding and truncation modes.

The vertical line at 23 fraction bits in both charts represents the vendor IP, highlighting the corresponding results. The vendor IP consumes less LUT than CuFF with the same bit-width, while it requires slightly more FF than CuFP(t). Both CuFP and the vendor IP exhibit the same number of cycles. The most significant difference is in DSP utilization, so the vendor IP uses 2 DSPs for floating-point addition, whereas CuFP does not utilize any DSPs.

Similar visualizations in Figure 6.6 demonstrate the resource utilization and cycles of CuFP with different fraction widths for rounding and truncation modes in the `mul` operation at a clock frequency of 200 MHz. The number of FFs in both modes is almost identical, exhibiting a linear increase with the growth of fraction bits. The utilization of DSPs increases with every other increment of fraction bits and remains nearly consistent between both modes, except at the 7-bit fraction width.



(a) CuFP(t) : Truncation mode

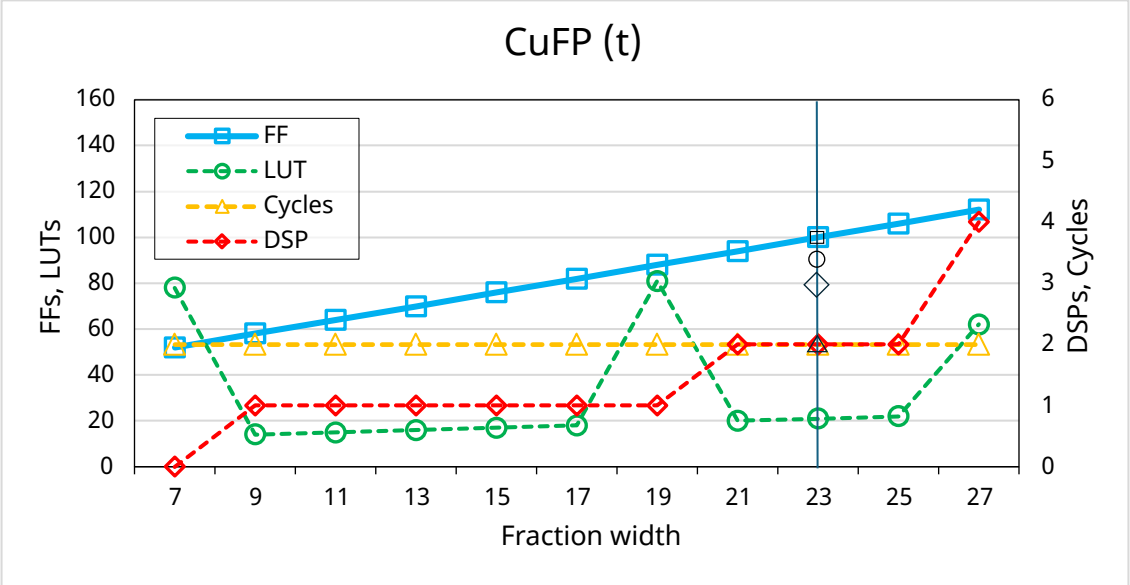


(b) CuFP(r) : Rounding mode

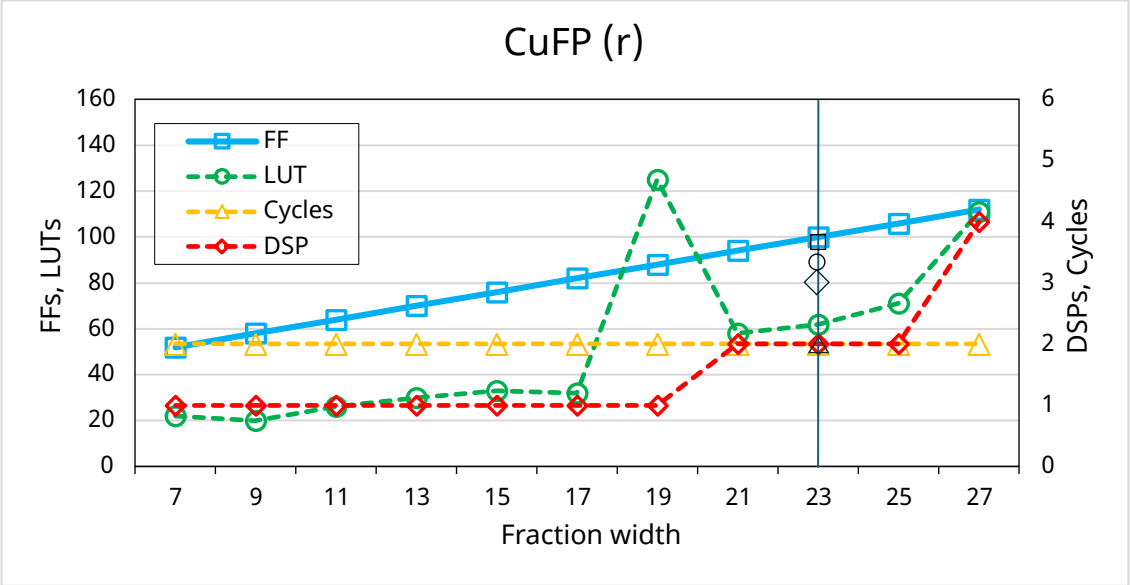
FIGURE 6.5 Resource utilization and cycles of CuFP (8, 23) for `sum` operation : fraction width varies from 7 to 27

Figure 6.6 reveals a notable peak in LUT utilization just before an increase in DSP usage, which is attributed to the oversizing of DSP operands during mantissa multiplication. Given that the input size of DSPs in the Virtex Ultrascale+ family is 18 × 27 [76], managing transitions beyond 18 fraction bits challenges the HLS tool in efficiently allocating logic. During these transitions, the tool initially allocates extra LUTs for fraction multiplication. As the fraction width increases further, it shifts to additional DSPs to handle larger inputs, normalizing LUT utilization. This demonstrates the HLS tool's balance between LUT and

DSP allocation to optimize resource use for the `mul` operation in CuFP.



(a) CuFP(t) : Truncation mode



(b) CuFP(r) : Rounding mode

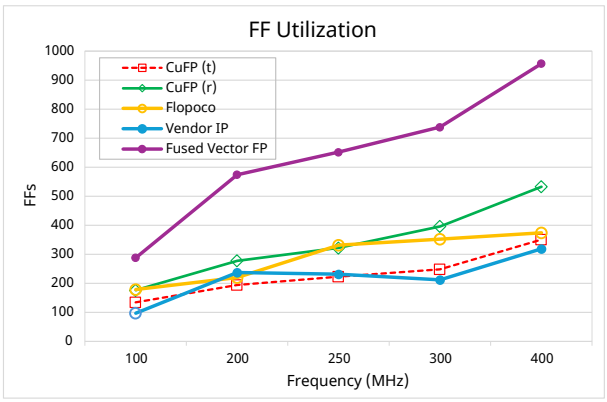
FIGURE 6.6 Resource utilization and cycles of CuFP (8, 23) for `mul` operation : fraction width varies from 7 to 27

Regarding the vendor IP for a single-precision floating-point multiplier, it consumes more LUTs and DSPs than CuFP in both rounding and truncation modes. The number of cycles

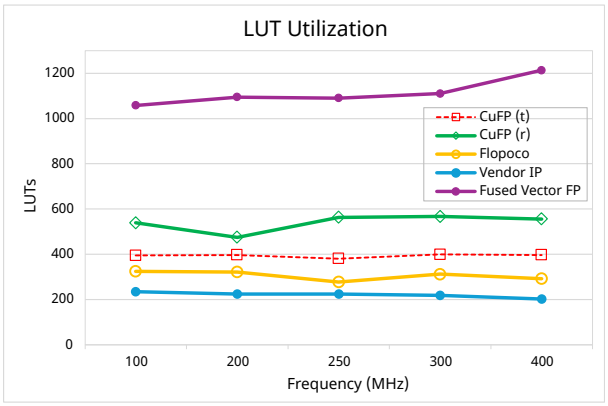
and FFs between the vendor IP and CuFP are identical. This indicates that while the vendor IP may provide a standard implementation, CuFP offers a heterogeneous data format and achieves similar performance with more efficient utilization of resources, particularly in terms of LUT and DSP consumption.

6.4.2 Comparative Analysis of CuFP

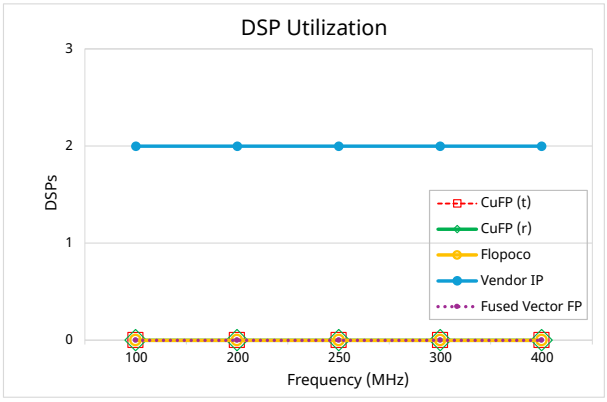
In this part of the evaluation, CuFP is compared to the existing floating-point vendor IP, Flopoco, and Fused Vector FP. In this experiment, the bit-width of input and output numbers is considered fixed and the same as the standard 32-bit floating-point. Figure 6.7 illustrates resource utilization trends for the `sum` operation at different clock frequencies. All the evaluated methods typically require more cycles to complete their processes as the clock frequency increases. Flopoco generally requires more cycles than the other methods. Correspondingly, they exhibit growing FF utilization for storing internal signals and passing them to subsequent cycles. The Fused Vector FP method consumes notably more FFs than the others, exhibiting a steeper rate of increase. The vendor IP and CuFP(t) show competitive FF consumption, while Flopoco and CuFP(r) utilize more FFs than the others when the clock frequency exceeds 200 MHz. In contrast, LUT consumption is not significantly affected by changes in clock frequency. The vendor's `sum` operation, Flopoco, Fused Vector FP, CuFP(t), and CuFP(r) require approximately 210, 300, 1100, 400, and 490 to 570 LUTs, respectively. The high LUT and FF utilization of Fused Vector FP may be attributed to a lack of specific optimization for FPGA-based solutions. None of the methods consume DSPs except for the vendor IP, which requires 2 DSPs.



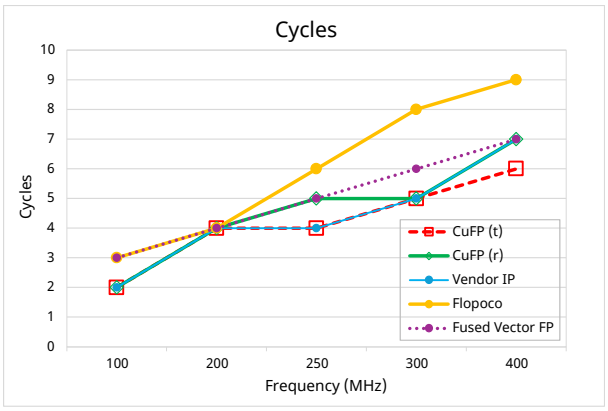
(a) FF Utilization



(b) LUT Utilization



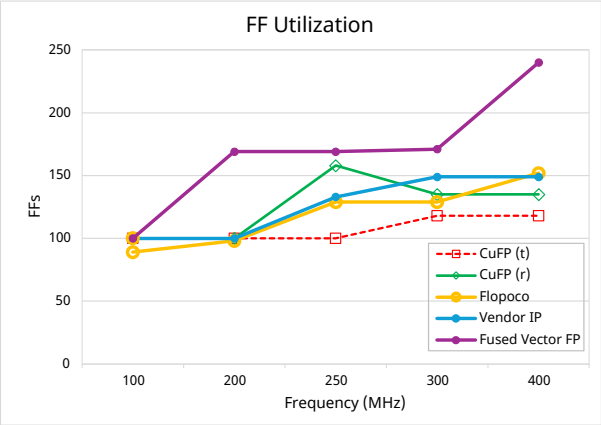
(c) DSP Utilization



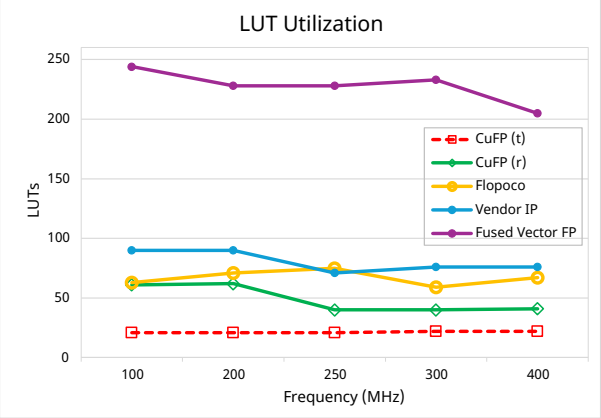
(d) Number of clock cycles

FIGURE 6.7 Comparing the resource utilization of `sum` operation for CuFP (8, 23), Flopoco, Fused Vector FP [50], and vendor IP as the target clock constraint is swept from 100 to 400 MHz

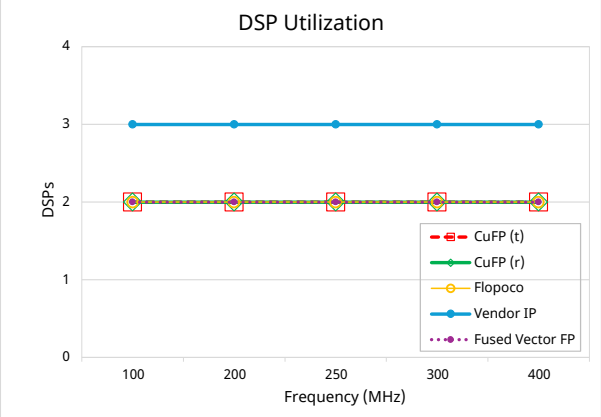
Figure 6.8 compares the mentioned methods for their `mul` operations. CuFP demonstrates superior performance in terms of clock cycles in both modes, requiring only 2 to 3 cycles. Similarly, FF consumption is competitive, ranging from approximately 95 to 160. CuFP(t) exhibits the lowest FF utilization across most clock frequencies. Regarding LUT utilization, CuFP(t) proves to be the most efficient method, consuming between 2×, 5×, and 11× fewer LUTs than the vendor IP, Flopoco, and Fused Vector FP, respectively. As observed with the `sum` operation, changing the clock frequency does not significantly impact the number of LUTs used. Finally, all methods for `mul` operations, except for the vendor IP, utilize 2 DSPs consistently across all clock frequencies. The vendor IP requires 3 DSPs regardless of the clock frequency.



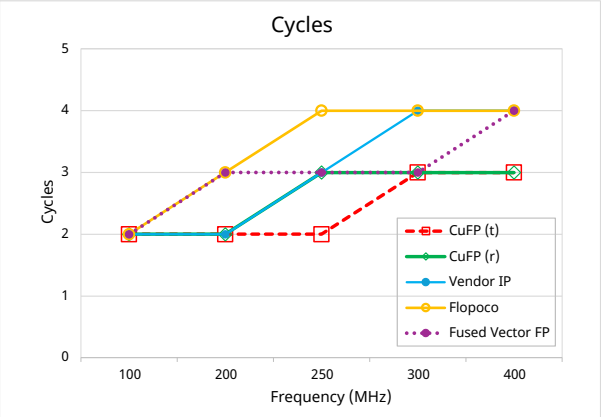
(a) FF Utilization



(b) LUT Utilization



(c) DSP Utilization



(d) Number of clock cycles

FIGURE 6.8 Comparing the resource utilization of `mul` operation for CuFP (8, 23), Flopoco, Fused Vector FP [50], and vendor IP as the target clock constraint is swept from 100 to 400 MHz

Table 6.3 presents a comparative analysis of the maximum clock frequencies, the number of cycles, and resource utilization for `sum` and `mul` operations across CuFP (both rounding and truncation modes), Vendor IP, and Flopoco. CuFP performs better regarding maximum clock frequency for both `sum` and operations compared to the Vendor IP and Flopoco. Specifically, CuFP(t) achieves the highest clock frequencies, reaching 436 MHz and 468 MHz for `sum` and `mul` operations, respectively. Regarding resource utilization, CuFP(t) consumes fewer FFs and LUTs than CuFP(r) and requires no DSPs for the `sum` operation, unlike the Vendor IP which uses 2 DSPs. Both CuFP modes efficiently utilize 2 DSPs for the `mul` operation, while the Vendor IP requires 3 DSPs.

TABLE 6.3 Comparing the maximum possible clock frequency of Sum and Mul operations for CuFP (8, 23) in two modes of rounding, Vendor IP (single-precision floating-point), and Flopoco (8, 23).

Operation	Variant	# of Cycles	Max Frequency (MHz)	Resource Utilization		
				DSP	LUT	FF
Sum	CuFP (r)	6	420	0	486	414
	CuFP (t)	6	436	0	410	314
	Vendor IP	6	375	2	219	277
	Flopoco	6	332	0	278	331
Mul	CuFP (r)	3	436	2	41	135
	CuFP (t)	3	468	2	22	118
	Vendor IP	3	284	3	71	133
	Flopoco	3	338	2	71	98

6.4.3 Evaluation of Dedicated Vector Operations

This subsection comprehensively evaluates the proposed dedicated vector operations in the CuFP library. The focus is on three critical operations : `vsum`, `dp`, and `mvm`.

Vector Summation

The resource utilization and performance results of the `vsum` operation presented in Table 8.2 show improvement of CuFP over the vendor IP, particularly in terms of clock cycle efficiency. CuFP consistently requires fewer clock cycles to complete operations. For instance, CuFP handles vector sizes with 4 to 32 elements using only 4 to 6 cycles, whereas the vendor IP requires 8 to 20 cycles for the same operations. This significant reduction in the number of cycles enhances processing speed and throughput considerably.

Moreover, CuFP achieves this clock cycle efficiency without relying on DSPs, which is a notable advantage. While the vendor IP consumes a substantial number of DSPs (ranging from 6 to 62 DSPs), CuFP operates efficiently without any DSPs. Although CuFP uses more LUTs and FFs, this trade-off is beneficial considering the reduced clock cycles and DSP independence. The clock cycle efficiency of CuFP improves computing performance while simultaneously providing flexibility in input vector size and data width.

TABLE 6.4 Comparing the resource utilization of `vsum` with different vector sizes for CuFP (8, 23) and vendor IP (single precision floating-point), at 200 MHz clock frequency.

Variant	**Vector Size**	**# Cycles**	**Resource Utilization**		
			DSP	**LUT**	**FF**
CuFP	4	4	0	1,071	441
	8	5	0	2,110	685
	16	5	0	3,675	1,277
	32	6	0	7,420	3,050
Vendor IP	4	8	6	701	709
	8	12	14	1,642	1,645
	16	16	30	3,522	3,513
	32	20	62	7,283	7,245

Dot-Product

Table 8.3 presents the resource utilization and performance results of the `dp` operation, and reveals significant improvement of CuFP over both the vendor IP and the Fused Vector FP method, particularly in clock cycle efficiency. CuFP consistently demonstrates superior clock cycles, completing a `dp` operation with fewer clock cycles than the vendor IP and Fused Vector FP across varying vector sizes. For example, at a vector size of 32, CuFP requires only 7 cycles, while the vendor IP needs 22 cycles and Fused Vector FP requires 14 cycles, resulting in significantly lower clock cycles for CuFP.

This clock cycle efficiency of CuFP translates to enhanced processing speed and throughput, making it highly suitable for time-sensitive applications. CuFP exhibits efficient resource utilization, particularly evident in linear growth in DSP usage with increasing vector size. While CuFP consumes more LUTs than the vendor IP, it uses significantly fewer LUTs and FFs than Fused Vector FP. Despite consuming more LUTs than the vendor IP, CuFP's efficient DSP and FF utilization and lower cycle count contribute to its overall resource efficiency and potential for faster performance.

The efficiency of CuFP in `dp` operation is rooted in its implementation using a generic `vsum`, as discussed earlier. By leveraging the efficient architecture of `vsum`, CuFP minimizes the number of cycles and maximizes DSP utilization, resulting in faster computation and lower latency across different vector sizes.

TABLE 6.5 Comparing the resource utilization of `dp` with different vector sizes for CuFP (8, 23) and vendor IP (single-precision floating-point), at 200 MHz clock frequency.

Variant	Vector Size	# Cycles	Resource Utilization		
			DSP	LUT	FF
CuFP	4	5	8	1,427	562
	8	5	16	2,629	1,085
	16	6	32	4,736	2,350
	32	7	64	9,117	4,831
Fused Vector FP [50]	4	6	8	3,247	968
	8	8	16	6,131	2,465
	16	9	32	12,240	5,521
	32	14	64	26,537	10,529
Vendor IP	4	10	18	1,057	1,095
	8	14	38	2,354	2,415
	16	18	78	4,947	5,051
	32	22	158	10,138	10,319

Matrix-Vector Multiplication

The resource utilization and performance results of the `mvm` operation are presented in Table 6.6. The efficiency of CuFP in `mvm` operations is demonstrated through its implementation leveraging the generic `dp` operation. The CuFP variant exhibits a consistent increase in resource utilization (DSP, LUT, and FF) with growing vector sizes while maintaining a moderate number of cycles. For instance, with a vector size of 4, CuFP uses 32 DSPs, 5230 LUTs, and 1831 FFs over 5 cycles. As the vector size increases to 32, the DSP usage scales to 2048, with 294,019 LUTs and 120,679 FFs over 7 cycles. This efficient scaling indicates CuFP's robust architecture, which is designed to handle larger computations effectively. Comparatively, the vendor IP displays a more significant increase in both resource utilization and the number of cycles. For a vector size of 4, the vendor IP requires 72 DSPs, 4240 LUTs, and 3960 FFs over 10 cycles. At a vector size of 32, resource utilization jumps to 5056 DSPs, 324,261 LUTs, and 297,720 FFs over 22 cycles. While the vendor IP uses fewer LUTs for smaller vector sizes, it becomes less efficient as vector sizes grow, with significantly higher DSP and FF usage and a larger number of cycles than CuFP.

TABLE 6.6 Comparing the resource utilization of `mvm` with different vector sizes for CuFP (8, 23) and vendor IP (single precision floating-point), at 200 MHz clock frequency.

Variant	Vector Size	# Cycles	Resource Utilization		
			DSP	LUT	FF
CuFP	4	5	32	5,230	1,831
	8	5	128	20,393	6,759
	16	6	512	80,974	29,320
	32	7	2,048	294,019	120,679
Vendor IP	4	10	72	4,240	3,960
	8	14	304	18,859	17,416
	16	18	1,248	79,188	72,836
	32	22	5,056	324,261	297,720

6.5 Discussion

The CuFP library offers several benefits and demonstrates significant improvements over traditional vendor IP cores regarding flexibility, resource utilization, and performance, particularly in `vsum`, `dp`, and `mvm` operations. Its primary advantage is its flexibility in defining custom floating-point formats. By allowing users to specify the number of bits for the exponent and mantissa, the library enables a more tailored approach to precision and dynamic range based on the application's specific needs. This customization is particularly valuable in applications where a balance between precision and resource utilization is critical, such as in FPGA-based scientific computing and real-time signal processing.

The experimental results indicate that CuFP achieves lower latency and better resource utilization than vendor-provided IP cores. For instance, the `vsum` operation shows a notable reduction in the number of DSP slices and LUTs used and a decrease in the number of clock cycles required for computation. These improvements are consistent across different vector sizes and operations, highlighting the efficiency of the CuFP library in handling large-scale computations. Suppose an application running at 200 MHz clock frequency, deals with floating-point numbers, and requires the multiplication of a 32×32 matrix by a 32-element vector. This task involves $32 \times 32 = 1024$ multiplications, which can potentially be executed in parallel, and $32 \times 31 = 992$ additions (as each of the 32 rows requires 31 additions) that can be done in the form of balanced tree additions for each row. If the application employs standard single-precision floating-point vendor IP cores, it would take 22 clock cycles as

outlined in Table 6.6. However, CuFP offers a dedicated `mvm` operation that needs only 7 cycles to do the same calculations. Furthermore, CuFP uses approximately 60% fewer DSPs, 10% fewer LUTs, and 60% fewer FFs, making it a highly efficient and more advantageous option for such applications.

6.5.1 Comparison with Existing Solutions

When compared to other customizable floating-point libraries, such as FloPoCo and Fused Vector FP, CuFP offers several enhancements. Using C++ templates and HLS directives allows for more efficient hardware mapping and optimization. Unlike the Fused Vector FP, CuFP's support for heterogeneous floating-point types and its integration with HLS tools streamline the development process, making it easier for developers to incorporate custom floating-point functionality into their low- and high-level designs. The CuFP library demonstrates superior resource utilization, especially in operations that involve extensive arithmetic computations. The library minimizes the required resources by optimizing the alignment, normalization, and rounding processes while maintaining high performance. This optimization is evident in the experimental results, where CuFP consistently uses fewer DSP slices and LUTs than vendor IP cores.

For instance, for the `vsum` operation at a vector size of 32 and clock frequency of 200 MHz, CuFP utilizes 7420 LUTs and 3,050 FFs compared to the vendor IP with 7283 LUTs and 7245 FFs, but the vendor IP requires significantly more DSPs and stages. The `vsum` operation in CuFP can add 32 single-precision floating-point numbers approximately 3x faster than the usual method using the vendor-provided IP cores thanks to elaborately customizing floating-point addition and eliminating excess processing steps in the middle like packing, unpacking, and normalizing. Given that the other vectorized operations of CuFP are implemented by leveraging `vsum`, they also benefit from resource efficiency and high-speed performance.

6.5.2 Challenges and Limitations

Despite its advantages, the CuFP library also presents some challenges. The complexity of designing and implementing custom floating-point operations can lead to increased development time and effort. Additionally, while the library offers significant performance improvements, the trade-off between resource usage and precision must be carefully managed to avoid potential issues in applications that require extremely high accuracy. Future work on the CuFP library could focus on several areas for further enhancement. One potential direction is the development of additional custom operations and optimizations for specific application domains, such as machine learning. Additionally, exploring the integration of CuFP with other

high-level synthesis tools and FPGA platforms could broaden its applicability and usability. Moreover, further research into the automatic optimization of custom floating-point parameters could help ease the development process and reduce the manual effort required for fine-tuning designs. Enhancing the library's support for mixed-precision arithmetic and expanding its compatibility with various FPGA architectures would also contribute to its versatility and performance.

6.6 Conclusions

This paper introduces CuFP, an efficient HLS library for custom floating-point representations using C++ templates, designed to enhance flexibility and reduce latency in FPGA applications. By allowing users to define heterogeneous floating-point types at compile time, CuFP offers a broader spectrum of precision options, surpassing traditional vendor IP cores. Experimental results demonstrate that CuFP consistently outperforms vendor IP cores, particularly in `vsum`, `dp`, `mvm` operations. CuFP's dedicated `mvm` operation, for example, significantly reduces clock cycles and hardware resources.

Compared to other customizable floating-point libraries, CuFP leverages C++ templates and HLS directives to achieve more efficient hardware mapping and optimization. This results in superior performance in arithmetic computations while minimizing resource usage. However, the design and implementation of custom floating-point operations may require additional development time and effort.

Future enhancements to CuFP should focus on expanding its capabilities by developing additional custom operations, integrating with other high-level synthesis tools, and supporting mixed-precision arithmetic. These improvements will further solidify CuFP's position as a versatile and high-performance tool for FPGA-based applications.

CHAPITRE 7 ARTICLE 3 : FPGA-BASED SIMULATION OF GRID-TIED CONVERTERS USING FREQUENCY-DEPENDENT NETWORK EQUIVALENT

Fahimeh Hajizadeh, Alireza Masoom, Tarek Ould-Bachir, and Jean-Pierre David

Publié dans : 16th International Conference on Power Systems Transients (IPST) (journal : Electric Power Systems Research)

Date de soumission : 6 janvier 2025

Abstract

This paper introduces a real-time simulation framework for grid-tied converters, implemented on field-programmable gate arrays (FPGAs). The framework incorporates a Frequency-Dependent Network Equivalent (FDNE) to reduce the original part of the circuit that is not directly under study into a frequency-dependent admittance model, enabling precise modeling of the power network's frequency-dependent dynamics while streamlining the onboard simulation and modeling process. The proposed framework is implemented on the Alveo U280 FPGA, achieving sub-microsecond latencies, low resource utilization, and high computational fidelity across various data types, including single-, double-precision, and customized floating-point formats. The numerical test and validation were conducted using a high-voltage power network that includes detailed models of transmission lines, loads, and a Static Synchronous Compensator (STATCOM), etc. Simulation results show strong alignment with reference models developed in the EMTP, achieving faster-than-real-time performance. These findings demonstrate the effectiveness of the proposed solution in delivering high-speed, resource-efficient, and scalable real-time simulations, providing a promising approach for testing and validating advanced control strategies in modern power systems.

Keywords

FPGA, FDNE, Real-time simulation, Electromagnetic simulation, STATCOM

7.1 Introduction

The integration of power converters in modern power systems has become essential to accommodate the growing adoption of distributed energy resources (DER) such as wind turbines, solar photovoltaics and charging stations for electric vehicles [87]. Power converters are essen-

tial for connecting renewable energy sources to the grid, facilitating efficient energy transfer, and improving grid stability through active and reactive power management [88, 89]. These converters connect various renewable energy sources to the power grid, facilitating cleaner energy consumption and addressing intermittency and fluctuating load demands [90–92]. However, the extensive use of power electronic devices, especially in grids with many DERs, poses distinct challenges to grid stability. This situation requires advanced control solutions to address potential power quality issues.

However, using power converters, such as voltage source converters (VSCs) and other inverter-based resources, presents challenges for grid stability. These devices can interact with conventional synchronous machines and other grid elements, causing oscillations, higher frequency change rates, and frequency overshoots [93, 94]. Moreover, control strategies to manage the dynamic performance of these converters must be meticulously designed to minimize harmonic distortion and maintain unity power factor for improved grid resilience [95]. As the use of such devices grows, it is essential to establish effective control methodologies to manage interactions with traditional grid infrastructure.

This paper presents a simulation framework for grid-connected converters to address these challenges, designed using field-programmable gate arrays (FPGAs). The framework integrates a Frequency-Dependent Network Equivalent (FDNE) model to accurately represent the frequency-dependent behavior of the grid. The framework achieves sub-microsecond latencies while preserving computational accuracy. A Static Synchronous Compensator (STATCOM) is employed as a test case to validate the effectiveness of the proposed approach.

The primary contributions of this work include :

— The development of two novel FDNE integration approaches based on state-space equations, enabling efficient real-time simulation of an FDNE-integrated STATCOM model on FPGA. These formulations leverage matrix-based computations, optimizing execution speed and numerical stability.

— The implementation of a resource-efficient FPGA framework, utilizing high-level synthesis (HLS) for FPGA programming and integrating Customized Floating-Point based (CuFP-based) arithmetic. CuFP allows customizable precision, balancing accuracy, and hardware efficiency to achieve optimal FPGA utilization.

— A demonstration that the proposed model achieves faster-than-real-time performance, significantly reducing simulation latencies. This capability positions the framework as a powerful tool for accelerating electromagnetic transient (EMT) simulation applications. 4) An in-depth analysis of resource utilization and computational trade-offs, emphasizing the role of the CuFP library in delivering optimal results.

The remainder of this paper is organized as follows : Section 7.2 provides an overview of the key concepts. Section 7.3 outlines the proposed implementation methodology. Section 7.4 describes the test case used to assess the proposed model's performance compared to the commercial tool EMTP®. Section 7.5 offers a comprehensive analysis of the FPGA implementation, focusing on latency, resource utilization, and accuracy. Finally, Section 7.6 presents the conclusions.

7.2 Background

7.2.1 FPGA-based Power System Simulation

FPGAs have been used in real-time simulation applications for many years, particularly in hardware-in-the-loop (HIL) configurations. The inherent parallelism of FPGAs allows for the simulation of complex power electronic circuits with small time steps, ensuring high fidelity and precise interfacing with physical controllers [55,96]. Beyond real-time simulation, research has extended the use of FPGAs to faster-than-real-time (FTRT) simulation. This approach allows for simulating the behavior of systems in less time than their actual operation, which is beneficial for offline analysis, optimization, and design validation [97,98]. The authors in [96] provide a comprehensive review of the current state of real-time simulation technologies for power systems. The study focuses on digital real-time simulation (DRTS) and HIL simulation, examining their evolution, computing capabilities, common features, hardware and software components, and solution methodologies across various simulator platforms. The work has demonstrated the simulation of power electronics systems with time steps in the range of a few microseconds. The paper [99] emphasizes the role of real-time simulation technologies in design, prototyping, testing, and teaching, categorizing applications by field, fidelity, and multiphysics aspects, with a focus on transmission and distribution systems. It highlights that the time-step in real-time simulation is critical and varies by application : microsecond-range steps are used for HVdc systems and EMT simulations, while millisecond-range steps are typical for phasor simulations. Paper [100] presents a wide-band multi-port system equivalent for real-time digital simulators, integrating a FDNE for high-frequency EMT and Transient Stability Analysis (TSA) for electromechanical transients. This approach enables accurate simulation of both fast and slow power system dynamics while reducing hardware costs. The multi-port equivalent can be directly connected to the system boundary, allowing TSA to run on a real-time platform. The achieved time-steps vary by simulation type, with 25–50 µs for EMT simulations and 1–2 ms for TSA solutions.

7.2.2 FDNE

FDNE aims to reduce the computational burden of Transient simulations of large networks are done by dividing the network under study into two zones : the study zone and the external zone. Assuming that the external zone has a minor impact on a given transient study occurring within the study zone. An FDNE model consists of a rational model [26], its coefficients are calculated to match the frequency response of the subnetwork to replace (external zone) for a finite frequency band as :

$$\mathbf{Y}(s) \cong \mathbf{Y}_{fitted}(s) = \mathbf{G}_0 + s\mathbf{E} + \sum_{k=1}^{n} \frac{\mathbf{R}_k}{s - p_k} \tag{7.1}$$

where $\mathbf{Y}(s)$ is a $p \times p$ frequency-dependent admittance matrix, and coefficients p_k and $\mathbf{R}_k$ are poles and residues matrices, respectively ; n denotes the number of poles of the model ; and $\mathbf{G}_0$ and $\mathbf{E}$ are constant matrices ($\mathbf{E}$ is typically a zero matrix).

The proposed method is sensitive to the quality of the rational fitting of the FDNE, as poor fitting can lead to inaccuracies in the simulation results.

The FDNE model can be expressed using state-space equations as follows :

$$\begin{bmatrix} \dot{\mathbf{x}}(t) \\ \mathbf{i}_F(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.2}$$

where $\mathbf{x}(t)$ is the FDNE state variable. The matrices $\mathbf{A}$ and $\mathbf{C}$ contain the FDNE model poles and residues, with dimensions $(pn) \times (pn)$ and $p \times (pn)$, respectively. The matrix $\mathbf{B}$ consists of ones and zeros, structured accordingly, with a size of $(pn) \times p$. The matrix $\mathbf{D}$ is the same as in Eq. 7.1, with dimensions $p \times p$. the input and output vectors $\mathbf{v}(t)$ and $\mathbf{i}_F(t)$ represent voltages and currents, respectively. The Backward Euler method is applied in solving the state-space equations, where the next time-step is computed using precomputed system matrices, reducing computational complexity while preserving accuracy.

$$\begin{bmatrix} \mathbf{x}(t + \Delta t) \\ \mathbf{i}_F(t) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.3}$$

where $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}_d$, and $\mathbf{D}_d$ are discrete versions of matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$, respectively.

7.2.3 FPGA-based FDNE Simulation

FPGA-based FDNE simulations have gained prominence for achieving ultra-low latency and high parallelism, enabling sub-microsecond time steps. Additionally, in [60], the potential of real-time FDNE models was explored on both CPUs and FPGAs, focusing on parallelizing the FDNE algorithm and proposing a partitioning method to optimize computations. Moreover, an FPGA-based FDNE model for accurate real-time simulation of aircraft power systems was presented in [3], employing FDNE models to precisely represent aircraft power cables within an HIL simulation environment. Our work achieves sub-microsecond time-steps for a more intricate test case, highlighting the capability of FPGA-based simulations to efficiently model complex FDNE systems with excellent accuracy and precision for both real-time and FTRT applications.

7.2.4 HLS FPGA Programming

HLS bridges the gap between software-oriented design and hardware-level implementation, offering a streamlined approach to developing complex systems [67]. By enabling developers to describe digital system behavior using high-level programming languages like C or C++, HLS tools simplify FPGA programming. Furthermore, HLS tools automate numerous optimization tasks, reducing manual effort and significantly accelerating the development process.

The CuFP library enhances FPGA resource efficiency while ensuring scalability and computational accuracy, making it a powerful tool for future advancements in grid-tied converter simulations [4]. This work demonstrates how CuFP leverages high-level descriptions to reduce computational latency, optimizing performance while maintaining precision and adaptability in complex scenarios.

7.3 Implementation Methodology

7.3.1 Brief Test Case Presentation

Fig. 7.1 illustrates the test circuit implemented in EMTP® and used for voltage regulation in electrical power systems. This model represents a 500 kV, 100 MVA STATCOM connected to a 500 kV bus, typically used to stabilize the voltage at a high-voltage transmission line. The STATCOM includes a two-level voltage source converter (VSC) block, power transformer, and associated control and protection systems.

The detailed two-level topology is used for the VSC, and the valve comprises one IGBT switch, two non-ideal (series and anti-parallel) diodes, and a snubber circuit, as shown in

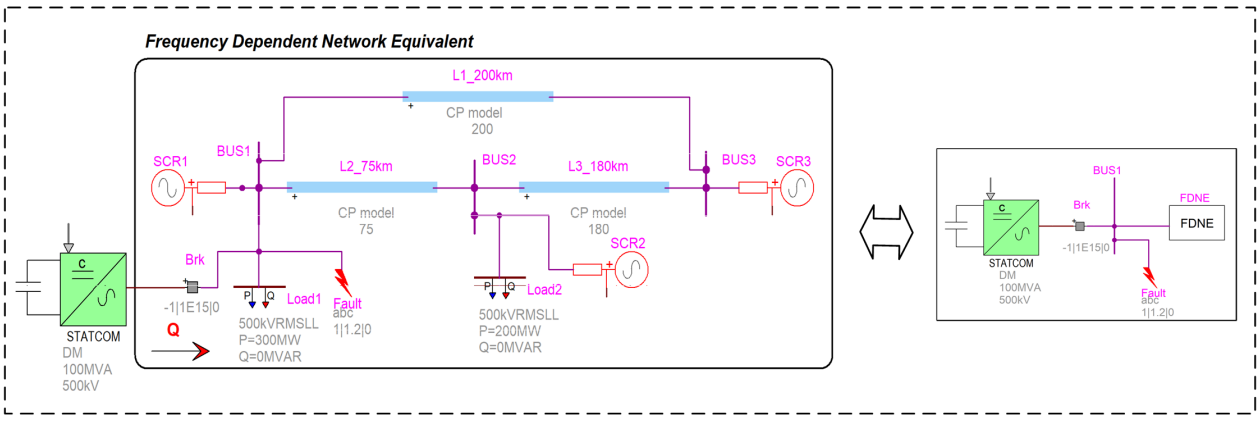


FIGURE 7.1 A +100 Mvar/-100 Mvar STATCOM implemented by two-level VSC.

Fig. 7.2. A non-ideal switch represents the diodes. The figure displays the configuration of the STATCOM and its integration within the power system, highlighting the components such as the coupling transformer, AC filters, and the control mechanism that manages the reactive power injection or absorption.

7.3.2 Integration of STATCOM and FDNE

This section examines two distinct approaches for integrating the STATCOM with the FDNE. While the equations presented in this work are explicitly derived for the STATCOM test case, the underlying methodology is broadly applicable and can be easily generalized to other grid-tied converter applications.

Approach 1

To characterize the dynamic behavior of the STATCOM, we start by defining its mathematical formulation as outlined in [55] :

$$\begin{bmatrix} \mathbf{i}_h(t+\Delta t) \\ \mathbf{i}_C(t) \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{11}^{\sigma} & \mathbf{H}_{12}^{\sigma} & \mathbf{H}_{13}^{\sigma} \\ \mathbf{H}_{21}^{\sigma} & \mathbf{H}_{22}^{\sigma} & \mathbf{H}_{23}^{\sigma} \end{bmatrix} \begin{bmatrix} \mathbf{i}_h(t) \\ \mathbf{v}_{in}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.4}$$

where $\mathbf{H}_{ij}$ are the matrices associated with switch combinations that result from the algebraic manipulations of the MANA matrix, the notation σ refers to the switch combination, $\mathbf{i}_h$ are history terms of the STATCOM, $\mathbf{v}_{in}$ internal voltage sources, $\mathbf{v}$ external voltage sources, and $\mathbf{i}_C$ the current drawn from the external sources (FDNE). The size of the matrix $\mathbf{H}$ depends on the number of history terms and the number of internal and external voltages. State update

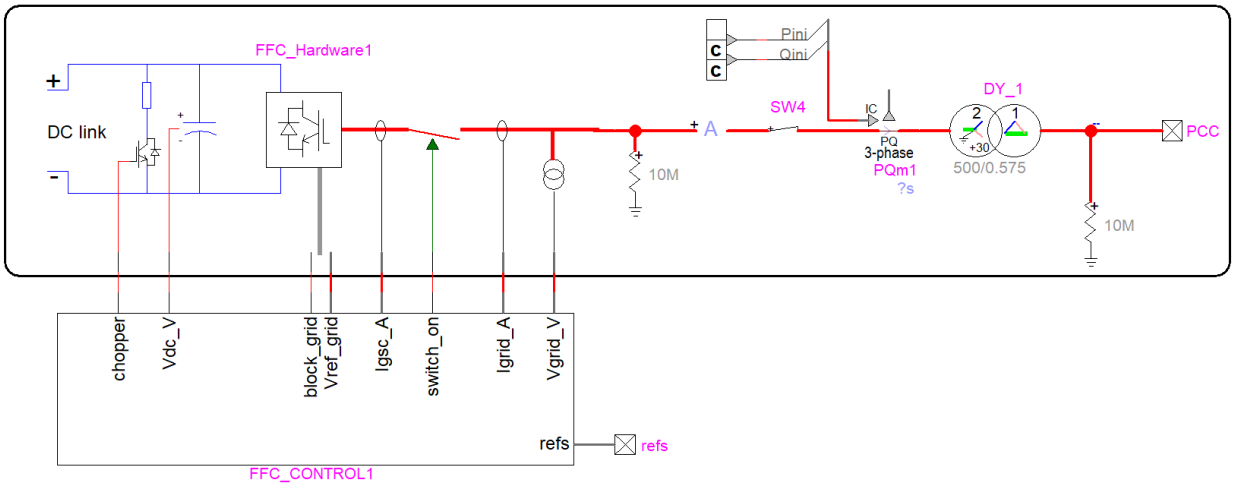


FIGURE 7.2 Detailed view of the STATCOM.

and output expressions in (7.4) can be separated, as shown in (7.5), (7.6) :

$$\mathbf{i}_h(t+\Delta t) = \begin{bmatrix} \mathbf{H}_{11}^{\sigma} & \mathbf{H}_{12}^{\sigma} & \mathbf{H}_{13}^{\sigma} \end{bmatrix} \begin{bmatrix} \mathbf{i}_h(t) \\ \mathbf{v}_{in}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.5}$$

$$\mathbf{i}_C(t) = \begin{bmatrix} \mathbf{H}_{21}^{\sigma} & \mathbf{H}_{22}^{\sigma} & \mathbf{H}_{23}^{\sigma} \end{bmatrix} \begin{bmatrix} \mathbf{i}_h(t) \\ \mathbf{v}_{in}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.6}$$

Same state update and output expressions separation can be applied for the FDNE in (7.3), we have (7.7), (7.8) :

$$\mathbf{x}(t+\Delta t) = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.7}$$

$$\mathbf{i}_F(t) = \begin{bmatrix} \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{v}(t) \end{bmatrix} \tag{7.8}$$

When the STATCOM and FDNE are integrated into the system, the following relationship is established : $\mathbf{i}_F(t) = -\mathbf{i}_C(t)$. Thus, by combining the (7.6), and (7.8), the following expression for computing the voltage is obtained :

$$\mathbf{v}(t) = \begin{bmatrix} \mathbf{W}_1^\sigma & \mathbf{W}_2^\sigma & \mathbf{W}_3^\sigma \end{bmatrix} \begin{bmatrix} \mathbf{i}_h(t) \\ \mathbf{v}_{in}(t) \\ \mathbf{x}(t) \end{bmatrix} \tag{7.9}$$

where

$$\begin{cases} \mathbf{W}_1^\sigma = -(\mathbf{H}_{23}^\sigma + \mathbf{D}_d)^{-1}\mathbf{H}_{21}^\sigma \\ \mathbf{W}_2^\sigma = -(\mathbf{H}_{23}^\sigma + \mathbf{D}_d)^{-1}\mathbf{H}_{22}^\sigma \\ \mathbf{W}_3^\sigma = -(\mathbf{H}_{23}^\sigma + \mathbf{D}_d)^{-1}\mathbf{C}_d \end{cases} \tag{7.10}$$

The following algorithm simulates the model in the time domain with a fixed time-step Δt for Approach 1.

Algorithm 1 Simulation Procedure for Approach 1

1: Precompute matrices $\mathbf{A}_d, \mathbf{B}_d, \mathbf{H}_{ij}^\sigma, \mathbf{W}_1^\sigma, \mathbf{W}_2^\sigma$, and $\mathbf{W}_3^\sigma$.
2: **for** each time point t **do**
3: Solve for $\mathbf{v}(t)$ using (7.9).
4: Optional : compute any output needed, e.g., using (7.6).
5: Update states of the STATCOM $\mathbf{i_h}(t)$ using (7.5).
6: Update states of the FDNE $\mathbf{x}(t)$ using (7.7).
7: **end for**

Approach 2

The purpose of Approach 2 is to reduce the computation latency at each time-point with a few additional precomputations. The idea here is to reduce matrix sizes in (7.9). Let's define $\boldsymbol{\xi}(t)$ as follows :

$$\boldsymbol{\xi}(t) = \mathbf{C}_d\mathbf{x}(t) \tag{7.11}$$

Hence allowing one to rewrite (7.9) as :

$$\mathbf{v}(t) = \begin{bmatrix} \mathbf{W}_1^\sigma & \mathbf{W}_2^\sigma & \mathbf{W}_4^\sigma \end{bmatrix} \begin{bmatrix} \mathbf{i}_h(t) \\ \mathbf{v}_{in}(t) \\ \boldsymbol{\xi}(t) \end{bmatrix} \tag{7.12}$$

where

$$\mathbf{W}_4^\sigma = -(\mathbf{H}_{23}^\sigma + \mathbf{D}_d)^{-1} \tag{7.13}$$

As we can rewrite (7.11) as follows :

$$\boldsymbol{\xi}(t+\Delta t) = \mathbf{C}_d\mathbf{x}(t+\Delta t) \tag{7.14}$$

Combining with (7.7), we have :

$$\boldsymbol{\xi}(t+\Delta t) = \begin{bmatrix}\mathbf{C}_d\mathbf{A}_d & \mathbf{C}_d\mathbf{B}_d\end{bmatrix}\begin{bmatrix}\mathbf{x}(t)\\ \mathbf{v}(t)\end{bmatrix} \tag{7.15}$$

The simulation algorithm becomes as follows :

Algorithm 2 Simulation Procedure for Approach 2

1: Precompute matrices $\mathbf{C_d A_d}, \mathbf{C_d B_d}, \mathbf{H}_{ij}^{\sigma}, \mathbf{W}_1^{\sigma}, \mathbf{W}_2^{\sigma}$, and $\mathbf{W}_4^{\sigma}$.
2: **for** for each time point t **do**
3: Solve for $\mathbf{v}(t)$ using (7.12).
4: Optional : Compute any output needed, e.g., using (7.6).
5: Update states of the STATCOM $\mathbf{i_h}(t)$ using (7.5).
6: Update states of the FDNE $\mathbf{x}(t)$ using (7.7).
7: Update variable $\boldsymbol{\xi}(t)$ using (7.15).
8: **end for**

The CPU is responsible for the precomputation of matrices ($\mathbf{A}_d, \mathbf{B}_d, \mathbf{H}_{ij}^{\sigma}, \mathbf{W}_1^{\sigma}, \mathbf{W}_2^{\sigma}$, and $\mathbf{W}_3^{\sigma}$) and ($\mathbf{C_d A_d}, \mathbf{C_d B_d}, \mathbf{H}_{ij}^{\sigma}, \mathbf{W}_1^{\sigma}, \mathbf{W}_2^{\sigma}$, and $\mathbf{W}_4^{\sigma}$), in approaches 1 and 2, respectively, ensuring efficient handling of computationally intensive operations. Once precomputed, these matrices are transferred to the FPGA, which performs real-time execution, solving system equations and updating states at each time step.

7.3.3 Latency Analysis for Approach 1 and Approach 2

Fig. 7.3 illustrates the datapath resulting from the integration of the STATCOM and the FDNE models within a unified simulation framework, according to Approach 1, whereas Fig. 7.4 shows the integration according to Approach 2. Fig. 7.5 illustrates the computational latency for Approach 1 and Approach 2. As shown in Fig. 7.5a, the maximum latency in Approach 1 is governed by two factors : (1) the latency of module $\mathbf{V}_{\mathrm{A1}}$, and (2) the maximum latency of the modules $\mathbf{I}$, $\mathbf{Y}$, and $\mathbf{X}$, which are executed in parallel. Since these modules operate concurrently, their collective latency is determined by the module with the highest computational delay. This approach ensures efficient parallelism within the constraints of the design.

$$\ell_{\textbf{Approach 1}} = \ell_{\mathbf{V}_{\mathrm{A1}}} + \max\left(\ell_{\mathbf{I}}, \ell_{\mathbf{Y}}, \ell_{\mathbf{X}}\right) \tag{7.16}$$

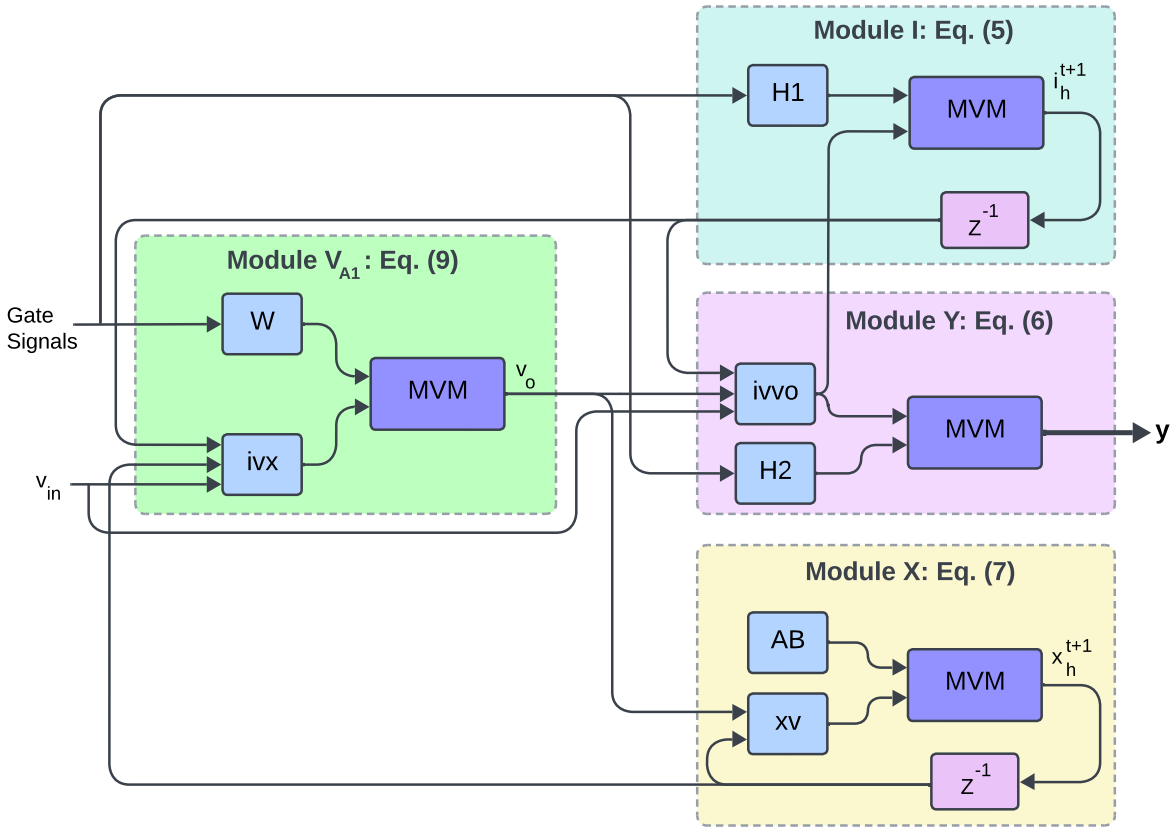


FIGURE 7.3 Integration of STATCOM and FDNE according to Approach 1. z^{-1} denotes a single time-step delay.

On the other hand, Fig. 7.5b illustrated the computational latency of Approach 2, which is determined by the latency of module $\mathbf{V}_{\mathrm{A2}}$ and the maximum latency of the modules **I**, **Y**, **X**, and **E**, which also operate in parallel.

$$\ell_{\textbf{Approach 2}} = \ell_{\mathbf{V}_{\mathrm{A2}}} + \max\left(\ell_{\mathbf{I}}, \ell_{\mathbf{Y}}, \ell_{\mathbf{X}}, \ell_{\mathbf{E}}\right) \tag{7.17}$$

A key observation is that the latency of module $\mathbf{V}_{\mathrm{A1}}$ is higher than $\mathbf{V}_{\mathrm{A2}}$. Hence, By introducing module **E** into the parallel pipeline, Approach 2 further distributes the workload, optimizing the overall latency.

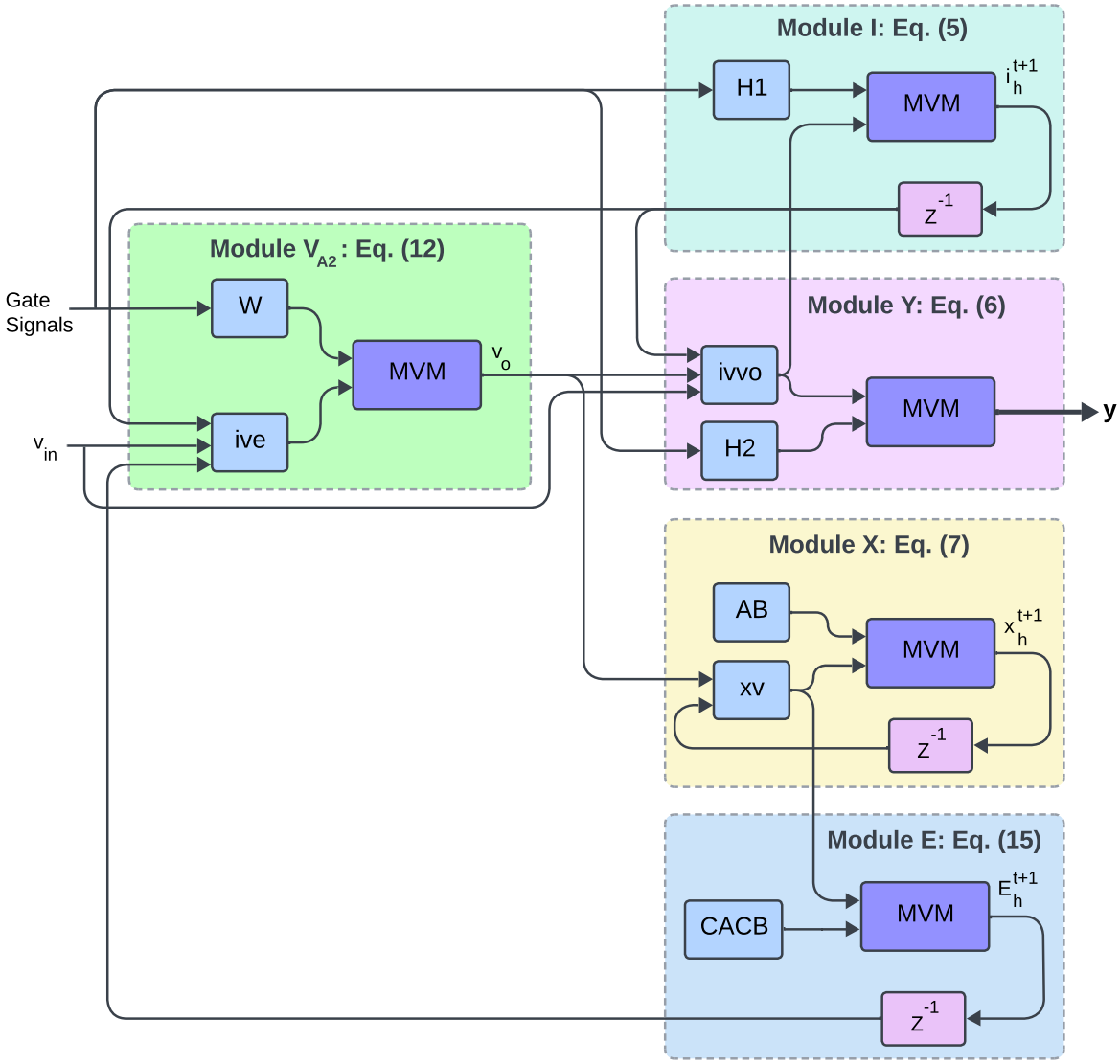
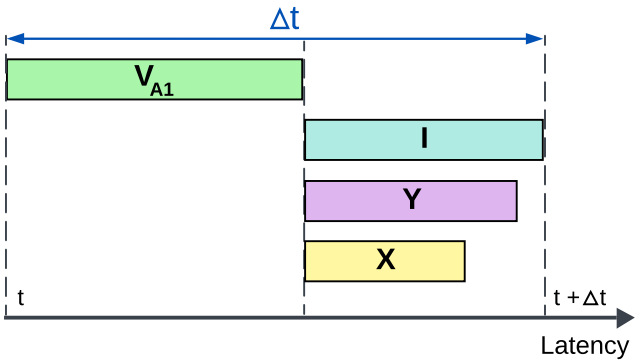


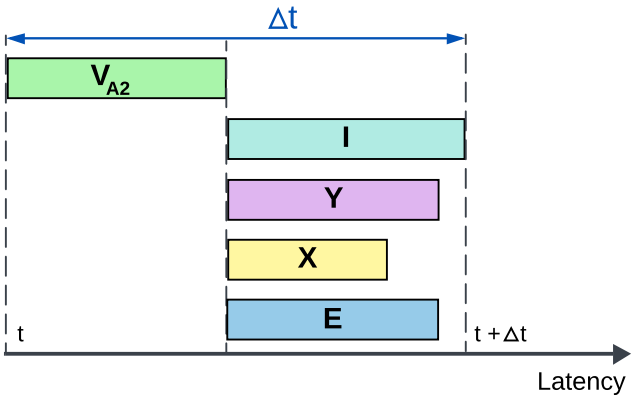
FIGURE 7.4 Integration of STATCOM and FDNE according to Approach 2. z^{-1} denotes a single time-step delay.

7.4 Test case

Fig. 7.1 illustrates the test circuit implemented in EMTP® and used for voltage regulation in electrical power systems. This model represents a 500 kV, 100 MVA STATCOM connected to a 500 kV bus, typically used to stabilize the voltage at a high-voltage transmission line. The figure displays the configuration of the STATCOM and its integration within the power system, highlighting the components such as the coupling transformer, AC filters, and the control mechanism that manages the reactive power injection or absorption. The FDNE model



(a) Approach 1



(b) Approach 2

FIGURE 7.5 Comparison of latency for different approaches.

contains an ideal current source to represent the steady-state conditions of the original circuit, connected in parallel to the state-space model described by Eq. 7.2. EMTP computes the parameters of FDNE by employing the vector fitting method combined with the Loewner-Matrix method [101] to determine the model order automatically. The frequency range spans from 1 Hz to 1 MHz, with a tolerance set to 1^{-6}, which results in 52 poles in the model. Therefore, the dimensions of the state-space matrices are as follows : A is 156×156, B is 156×3, C is 3×156, and D is 3×3.

For a dynamic response of STATCOM, a three-phase fault occurs on BUS 1 at $t = 1$ s and lasts for 200 ms. During the fault, the voltage at Bus 1 decreases from the reference voltage, $V_{\text{ref}} = 1$ pu. The STATCOM reacts to the event by generating the reactive power to increase the bus voltage.

7.5 Experimental Results

7.5.1 Simulation Accuracy

To validate the accuracy and fidelity of the proposed implementation, the results are compared with a reference model developed in EMTP®. Fig. 7.6a presents the superimposed phase currents at the receiving end, as computed by the proposed implementation and the EMTP® model, for the 0.9 to 1.3 seconds of the simulation. Detailed close-up views at critical events are shown in Fig. 7.6b and 7.6c, illustrating the system's behavior during fault initiation and clearing, respectively. As observed, the proposed implementation perfectly matches the reference model, confirming its high accuracy and reliability.

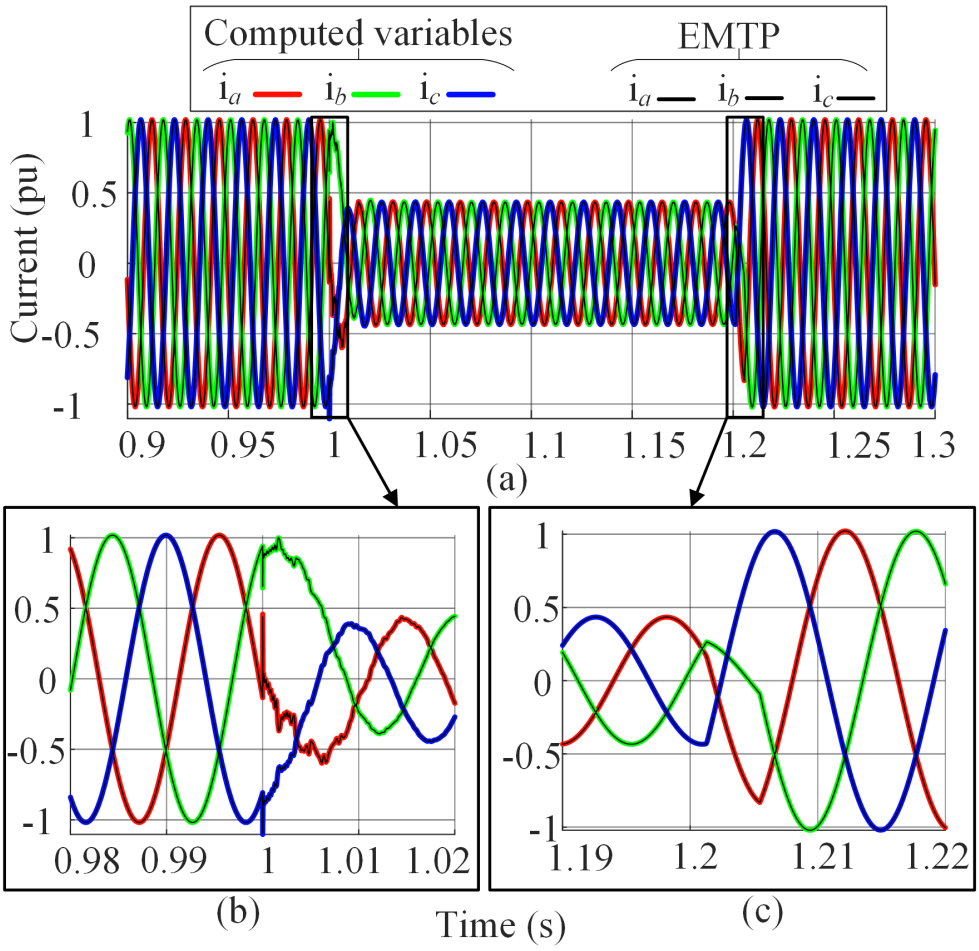


FIGURE 7.6 Comparison of the computed current with the reference current from the EMTP® model; (a) Close-up view of phase-currents during fault initiation; and (b) Close-up view of phase-currents during fault clearing.

7.5.2 Area occupation and speed performance

This section details the experimental results of implementing the proposed model on an Alveo U280 FPGA using Vitis HLS 2023.2. The selected FPGA platform, the xcu280-fsvh2892-2L-e, enables high-speed, real-time operation, with results reported at RTL simulation level, after post-routing to reflect actual hardware performance rather than simulation estimates. The offline simulations are carried out on EMTP® v4.5 to validate the FPGA implementation. The simulations are run on a laptop with an 11th Gen Intel i9-11950H processor and 64 GB DDR5 RAM.

The FDNE model size directly impacts FPGA resource utilization and execution time. As the number of poles in the admittance matrix increases, the state-space representation expands, leading to a larger system matrix. This results in higher memory usage and increased computational load due to additional state updates, affecting occupation and execution time. However, since our FPGA implementation leverages parallel computation and precomputed matrices, the latency increase remains moderate.

The proposed model is evaluated based on execution cycles, FPGA resource utilization, and accuracy. The model operates at a clock frequency of 250 MHz, and its performance is assessed using different data types : single, double, and customized floating-point formats provided by the CuFP library [4]. For the CuFP data type, three different configurations are chosen for comparison. In this library, the notation CuFP($w_e,\ w_m$) represents a floating-point number with an exponent bit width of w_e and a mantissa bit width of w_m. To enable meaningful comparisons with IEEE 754 single- and double-precision formats, CuFP (8, 24) and CuFP (11, 53) were specifically selected for this study. While fixed-point arithmetic could reduce FPGA resource usage, particularly for DSPs and LUTs, it requires careful bit-width optimization to avoid numerical instability due to its limited dynamic range. This is particularly challenging in power system simulations, where wide dynamic ranges are common.

To assess the accuracy of the proposed model, the 2-norm relative error ($\mathrm{RE}_{\text{2-norm}}$) is employed as the primary metric. This error metric, shown in (7.18), quantifies the relative discrepancy between the computed output from FPGA (out_{fpga}) and the reference output from EMTP® (out_{emtp}). Specifically, the $\mathrm{RE}_{\text{2-norm}}$ is calculated as follows :

$$\mathrm{RE}_{\text{2-norm}}(\%) = \frac{\left\| out_{\text{emtp}} - out_{\text{fpga}} \right\|_2}{\left\| out_{\text{emtp}} \right\|_2} \times 100 \tag{7.18}$$

where out_{emtp} represents the reference output and out_{fpga} represents the computed output. The symbol $\left\| out \right\|_2$ represents the 2-norm of the output.

Table 7.1 presents a comprehensive analysis of the performance metrics, including resource utilization and accuracy, of the proposed model for Approach 1 and Approach 2, evaluated across various data types. Key parameters such as latency, FPGA resource usage (BRAM, DSPs, LUTs, and Registers), and accuracy are compared.

TABLE 7.1 Comparison of the proposed model's performance in terms of number of cycles, resource utilization, and accuracy for Approach 1 and Approach 2.

Approach	**Data Type**	**Frequency (MHz)**	**# of Cycles**	**Latency (ns)**	**BRAM**	**DSPs**	**LUTs**	**Registers**	**RE_2-norm (%)**
Approach 1	Single-Precision Floating-Point	250	145	580	260 (6.44%)	470 (5.20%)	72,232 (5.57%)	86,106 (3.32%)	7.82×10^{-2}
	Double-Precision Floating-Point	250	180	720	403 (9.99%)	777 (8.43%)	163,013 (12.57%)	206,860 (7.98%)	1.03×10^{-2}
	CuFP (8, 24) [4]	250	102	408	254 (6.30%)	343 (3.80%)	122,036 (9.36%)	98,478 (3.78%)	8.7×10^{-2}
	CuFP (8, 34) [4]	250	116	464	300 (7.45%)	671 (7.44%)	182,332 (13.99%)	141,551 (5.43%)	5.07×10^{-2}
	CuFP (11, 53) [4]	250	139	556	326 (8.09%)	756 (10.61%)	227,619 (17.35%)	183,480 (6.75%)	1.08×10^{-2}
Approach 2	Single-Precision Floating-Point	250	123	492	263 (6.52%)	874 (9.68%)	90,303 (6.92%)	121,277 (4.65%)	7.81×10^{-2}
	Double-Precision Floating-Point	250	157	628	486 (12.05%)	1,676 (18.57%)	218,044 (16.72%)	270,270 (10.36%)	1.03×10^{-2}
	CuFP (8, 24) [4]	250	78	312	264 (6.61%)	374 (4.14%)	116,534 (8.94%)	87,178 (3.23%)	8.68×10^{-2}
	CuFP (8, 34) [4]	250	86	344	359 (8.92%)	780 (8.64%)	185,911 (14.26%)	140,549 (5.39%)	5.06×10^{-2}
	CuFP (11, 53) [4]	250	107	428	335 (8.31%)	957 (10.61%)	226,217 (17.35%)	175,961 (6.75%)	1.07×10^{-2}

In Approach 1, in the case of single-precision floating-point, the model achieves a latency of 580 ns and an RE_2-norm of 0.0782%. This represents a relatively low latency but with limited accuracy. CuFP (8, 24) shows a promising alternative, achieving a latency of 408 ns with slightly higher accuracy, offering comparable accuracy to single-precision while using fewer resources. This demonstrates the efficiency of CuFP (8, 24) in scenarios where resource constraints and latency are critical. CuFP (8, 34) configuration provides an even higher accuracy, as it uses more bits for the mantissa. However, this comes with an increase in latency and higher resource utilization. This configuration is suitable for applications where

accuracy is more critical than latency, but the higher resource requirements must be considered when working within resource-constrained environments. Finally, the double-precision floating-point implementation comes with a higher latency of 720 ns and higher resource consumption. In contrast, CuFP (11, 53) achieves a similar level of accuracy while maintaining a lower latency of 556 ns, and fewer resources than double-precision. This data type balances performance and efficiency, enabling fast computation with minimal resources, making it ideal for real-time simulation.

Moving to Approach 2, the single-precision floating-point implementation offers a latency of 492 ns and an RE_2-norm of 0.0781%. Resource utilization includes 263 BRAMs, 874 DSPs, 90,303 LUTs, and 121,277 registers. Compared to Approach 1, the latency is reduced, but the resource utilization is higher, reflecting the increased demand on resources in this approach. For double-precision floating-point, the latency is 628 ns and is reduced compared to Approach 1. However, the area is more than in Approach 1. When CuFP (8, 24) is utilized, the latency drops to 312 ns, with an RE_2-norm of 0.0868%, making it a highly efficient choice for applications requiring low latency and moderate accuracy. This configuration uses 264 BRAMs, 374 DSPs, 116,534 LUTs, and 87,178 registers, showcasing significant latency and resource efficiency improvements over floating-point implementations. For those applications that need more accuracy, the CuFP (8, 34) configuration can be a better option rather than CuFP (8, 24). This configuration offers an improved accuracy of 0.0506%, with a latency of 344 ns. Although the increased accuracy comes with higher resource utilization, the configuration offers a good balance between precision and performance. Similarly, CuFP (11, 53) achieves a latency of 428 ns, closely matching the accuracy of double-precision floating-point with an RE_2-norm of 0.0107%. Moreover, it consumes fewer resources than the double-precision implementation, demonstrating a balance between accuracy and resource utilization.

The proposed model demonstrates efficient resource utilization, low latency, and high accuracy across different data types and configurations. In particular, approach 2 with CuFP configurations offers a promising trade-off between latency, accuracy, and resource efficiency, making it an excellent choice for real-time applications. Additionally, the flexibility of the CuFP library allows users to customize the floating-point configuration to meet specific needs, providing enhanced control over performance and resource usage. The reduction in latency in Approach 2, however, comes at the expense of increased resource utilization. The additional computational resources required to support the expanded parallel pipeline result in a larger hardware footprint. This trade-off reflects a deliberate design choice to prioritize latency reduction over resource constraints. The comparative analysis highlights the benefits of leveraging parallelism to optimize latency. Approach 2 significantly reduces computational delay,

making it suitable for latency-sensitive applications, although with higher area requirements. The choice between these approaches depends on the specific application requirements and the hardware constraints of the target system. For applications where accuracy is the priority, CuFP (11, 53) or double-precision floating-point should be used to minimize numerical errors. If latency is the primary concern, particularly for real-time control applications, configurations in the second approach, such as CuFP (8, 24), provide a good balance between resource efficiency and numerical precision, significantly reducing execution time while maintaining acceptable accuracy. CuFP (8, 34) serves as a middle ground, offering improved accuracy over CuFP (8, 24) while keeping latency lower than full double-precision arithmetic. These results demonstrate that CuFP gives the flexibility to balance precision and computational efficiency, making it well-suited for various power system applications.

7.5.3 Speed-up Performance

In real-time simulation, a system is considered real-time if its execution time per step does not exceed the simulation time step (Δt). In our work, the latency per time step is in the nanosecond range (as shown in Table 7.1), significantly faster than the required microsecond-level time steps for power electronics simulations. The results highlight that the implementation achieved nanosecond-level latencies, making it well-suited for real-time applications. In comparison, the reference model, developed using EMTP® and executed on a system with the previously described configuration, was evaluated for specific time points, e.g. 50,000 time-points, with a fixed time-step. Under these conditions, the reference model exhibited a latency of 6.0625 seconds. By contrast, the proposed model, executed for the same number of time points, completed the task in just 24.6 ms using the single-precision data type, as detailed in (7.19).

$$\text{Total latency} = 492 \times 10^{-9} \times 50,000 = 24.6 \text{ ms} \tag{7.19}$$

This achievement represents a performance improvement of approximately 246 times compared to the reference model, highlighting that the proposed implementation is suitable for real-time applications and capable of operating faster than real-time.

7.6 Conclusions

This paper presented an efficient FPGA-based real-time simulation framework for grid-connected converters, integrating a STATCOM model with an FDNE approach. The proposed implementation was validated against a high-fidelity reference model developed in

EMTP®, demonstrating strong agreement in waveform accuracy and achieving minimal relative error across various floating-point formats. The results confirm the effectiveness of the proposed framework in replicating dynamic system behaviors with high precision. The FPGA implementation, tested on the Alveo U280 platform, showcases the potential for scalable and efficient real-time simulation systems. With sub-microsecond latencies and low resource utilization, the model is well-suited for integration into real-time control systems for power networks, where speed and accuracy are critical. Notably, the proposed approach achieved a remarkable 246 times speed improvement compared to the reference model, demonstrating its capability to perform faster-than-real-time simulations without compromising accuracy. Furthermore, the flexibility of the CuFP library enables future adaptations for different precision requirements and hardware configurations. Future work will focus on extending the methodology to multi-converter systems, such as multiple STATCOMs or active power filters, to assess scalability and performance trade-offs. Additionally, HIL validation will be conducted to confirm real-time execution in practical scenarios. Another research direction is adaptive precision selection, where CuFP bit-widths dynamically adjust based on system operating conditions to optimize computational efficiency further.

CHAPITRE 8 ARTICLE 4 : ENHANCING CUFP LIBRARY WITH SELF-ALIGNMENT TECHNIQUE

Fahimeh Hajizadeh, Tarek Ould-Bachir, and Jean-Pierre David

Publié dans : Computers

Date de parution : 24 mars 2025

Abstract

High-Level Synthesis (HLS) tools have transformed FPGA development by streamlining digital design and enhancing efficiency. Meanwhile, advancements in semiconductor technology now support the integration of hundreds of floating-point units on a single chip, enabling more resource-intensive computations. CuFP, an HLS library, facilitates the creation of customized floating-point operators with configurable exponent and mantissa bit widths, providing greater flexibility and resource efficiency. This paper introduces the integration of the self-alignment technique (SAT) into the CuFP library, extending its capability for customized addition-related floating-point operations with enhanced precision and resource utilization. Our findings demonstrate that incorporating SAT into CuFP enables the efficient FPGA deployment of complex floating-point operators, achieving significant reductions in computational latency and improved resource efficiency. Specifically, for a vector size of 64, CuFPSAF reduces execution cycles by 29.4% compared to CuFP and by 81.5% compared to vendor IP while maintaining the same DSP utilization as CuFP and reducing it by 59.7% compared to vendor IP. These results highlight the efficiency of SAT in FPGA-based floating-point computations.

Keywords

floating point ; high-level synthesis (HLS) ; FPGA ; customized floating point ; self-alignment technique (SAT) ; dot product (DP)

8.1 Introduction

Advances in semiconductor technology have enabled the development of highly dense devices capable of hosting tens to hundreds of floating-point operators on a single chip. Modern field-programmable gate arrays (FPGAs) exemplify this trend, offering substantial computational resources for applications requiring high performance and flexibility [86, 102]. However, leve-

raging these devices for computationally intensive tasks, such as floating-point dot product operations, remains challenging due to the trade-offs between precision, hardware area, and latency.

In [4], the authors introduced the CuFP library, a customizable floating-point library compatible with High-Level Synthesis (HLS) tools. CuFP allows developers to define floating-point formats with tailored precision by specifying the bit widths of the exponent and mantissa, enabling application-specific optimizations. While this customization significantly enhances flexibility in balancing precision and resource usage, it can also introduce challenges in computational performance for key operations, such as the dot product (DP) [103]. In large-scale systems and high-precision configurations, increasing the bit widths of the exponent and mantissa often extends the critical path in arithmetic operations, leading to longer delays and reduced throughput. To address these challenges, hardware implementations are typically optimized to minimize critical path delays. To some extent, leveraging techniques such as pipelining, loop unrolling, and resource partitioning can help mitigate these issues while preserving the precision benefits provided by CuFP. However, there are scenarios where these traditional optimization techniques fall short.

To address this limitation, we propose an enhanced the CuFP library (CuFPSAF) that integrates a Self-Alignment Format (SAF) [1] into the CuFP library to create a scalable, resource-efficient floating-point arithmetic framework for FPGA-based computing. SAF achieves significant reductions in latency and hardware area by keeping the alignment of the mantissa and exponent and removing unnecessary packing/unpacking stages outside of the critical path. This approach ensures that alignment operations do not directly impact the execution time of arithmetic computations. However, it has not been designed for seamless integration into an HLS framework. By bridging the gap between CuFP's configurability and SAF's efficiency, CuFPSAF provides a novel solution that significantly reduces the latency and hardware area required for floating-point operations across various precision formats, including simple, double, and quadruple precision. Integrating CuFPSAF into other projects is simple, as users can easily add a header file, ensuring smooth integration and a user-friendly implementation. Its open-source and platform-independent design fosters transparency, simplifies debugging, and ensures high performance `https://github.com/FahimeHajizadeh/Custom-Float-HLS` (accessed on 31 January 2025).

In this work, we also present some well-known operators, such as vector summation, dot product, and matrix-vector multiplication, based on the proposed CuFPSAF library and evaluate their performance in terms of latency and area occupation. The results demonstrate that CuFPSAF enables more efficient implementation of complex floating-point operations,

particularly when applied to computationally intensive tasks. Additionally, we highlight the benefits of a fused-path approach to enhance performance while maintaining accuracy.

The remainder of this paper is organized as follows : Section 8.2 presents an overview of floating-point numbers, the CuFP library, and some acceleration techniques. Section 8.3 details the CuFPSAF design and development flow. Section 8.4 provides a comprehensive evaluation of the proposed approach. Section 8.5 disscuses more about the proposed design. Finally, Section 8.6 concludes the paper and outlines potential avenues for future research.

8.2 Background

8.2.1 Floating-Point Format

Floating-point arithmetic is a fundamental aspect of numerical computation, enabling efficient representation and manipulation of a wide range of real numbers. The IEEE-754 standard is a globally recognized and extensively utilized framework for representing floating-point numbers [37]. This standard defines a format for expressing real numbers in a way that balances precision and dynamic range, enabling accurate and efficient computation in scientific, engineering, and data-intensive applications. Floating-point numbers under this standard are expressed in the form :

$$x = (-1)^s \cdot m \cdot 2^e \tag{8.1}$$

where s is the sign bit, m is the normalized mantissa m $\in [1, 2)$, and e is the exponent, typically stored in a biased form. This format allows efficient representation of a wide range of numbers, accommodating both very large and extremely small values. As shown in Figure 8.1, the IEEE-754 standard defines several precision levels to accommodate various computational needs : half-precision (16 bits), single-precision (32 bits), double-precision (64 bits), and quadruple-precision (128 bits).

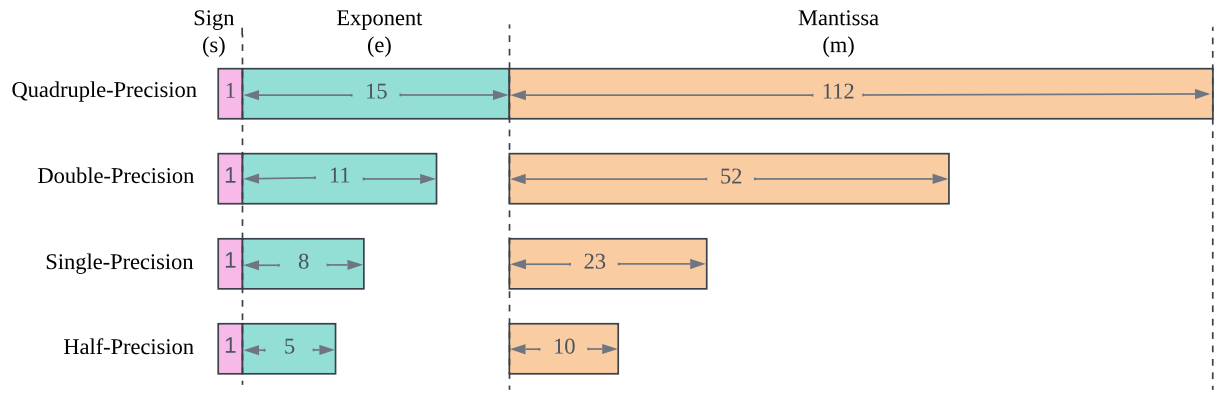


FIGURE 8.1 Several precision levels of IEEE-754 standard.

8.2.2 Challenges in Floating-Point Operations

Floating-point arithmetic plays a crucial role in many computational applications. However, performing floating-point operations, especially in high-performance contexts, presents several challenges. One of the primary concerns is the computational complexity and resource demands associated with these operations. Floating-point calculations often involve multiple steps, such as exponentiation and mantissa normalization, which can be resource-intensive. This complexity is further amplified in hardware implementations, particularly in systems like FPGAs, where high precision requirements can significantly increase the area, power consumption, and processing time. For applications such as dot products or matrix multiplications, these operations can become a bottleneck, limiting the overall performance and efficiency of the system.

Moreover, standard floating-point formats, which are defined by IEEE 754, are not always well suited for custom hardware applications. While IEEE 754 provides a standardized approach to floating-point representation, it is designed for general-purpose use and may not meet the specific needs of certain applications [104]. For example, standard formats, such as 32-bit or 64-bit floating-point formats, can be inefficient when precision is required only within a specific range. These formats may waste hardware resources, leading to suboptimal utilization of the available logic, increased latency, and unnecessary power consumption. This inefficiency can be particularly problematic in high-performance or real-time systems, as it introduces unnecessary overhead that limits the potential for optimization [82, 83]. Therefore, there is a growing need for customizable floating-point representations that offer greater flexibility in balancing precision, resource utilization, and computational efficiency tailored to the unique requirements of the target application [39, 40, 46, 48, 50, 105]. In addition, high

performance in FPGA-based floating-point computing can be achieved by eliminating unnecessary packing/unpacking stages as well as removing the intermediate stages [4,42,106,107]. Template HLS (THLS) [46] is an HLS library that provides a high-level approach to implementing custom floating-point operations using C++ templates. This framework enables the selection of exponent and fraction bit widths at compile time. Although THLS delivers a distinctive solution for both simulation and synthesis, its scope is largely limited to fundamental operations. TrueFlaot [48] is another work that is combined with the open-source HLS framework Bambu [49]. Moreover, the authors in [50] propose a parameterized, fused multi-term floating-point dot product architecture specifically optimized for high-level synthesis. While this architecture showcases significant advancements, it does not support heterogeneous formats, which reduces its versatility and applicability in scenarios requiring mixed precision computations.

8.2.3 Role of High-Level Synthesis (HLS)

High-Level Synthesis (HLS) is a powerful tool that bridges the gap between software design and hardware implementation. It allows developers to describe hardware behavior using high-level programming languages such as C, C++, or OpenCL rather than dealing with low-level hardware description languages like VHDL or Verilog [67]. This abstraction simplifies the design process and significantly accelerates development cycles, enabling more complex systems to be implemented with greater ease.

The primary benefit of HLS is its ability to translate high-level code into hardware descriptions that can be synthesized into FPGA designs [30, 108]. By using high-level languages, developers can focus more on the algorithmic aspects of their designs, leaving the hardware optimization to the HLS toolchain. HLS tools analyze the code and generate hardware circuits that best match the desired functionality while also optimizing for performance, area, and power consumption. This streamlines the process of creating custom hardware accelerators tailored to specific computational tasks, such as floating-point operations.

8.2.4 Overview of the CuFP Library

The CuFP library, presented in [4], is a specialized collection of floating-point operators designed to address the limitations of standard IEEE 754 formats, offering increased flexibility for high-performance and resource-constrained applications. The core idea behind CuFP is to enable the customization of floating-point formats, allowing the user to adjust key parameters such as the mantissa and exponent widths. This ability to tailor floating-point precision is crucial for applications that require specific trade-offs between computational precision and

hardware resource utilization.

CuFP provides a set of high-level functions that can be synthesized onto hardware platforms, such as FPGAs, using HLS tools. These functions are optimized to ensure efficient hardware utilization and fast execution times, especially in computationally intensive operations such as dot products, matrix-vector multiplications, and other numerical computations. By offering fine-grained control over the floating-point representation, CuFP allows for more efficient use of FPGA resources and reduces latency while maintaining adequate precision for the given application.

A distinguishing feature of CuFP is its flexibility in balancing precision and resource consumption. The library supports dynamic configurations where the user can define the number of bits used for the mantissa and exponent, making it adaptable to various levels of precision requirements. This customizability ensures that CuFP can meet the needs of diverse applications, from embedded systems to high-performance computing. In the context of real-time simulations, CuFP is particularly beneficial as it enables the acceleration of floating-point operations, allowing complex calculations to be performed with minimal delay. The library's ability to seamlessly interact with HLS tools further facilitates its integration into FPGA-based systems, making it an ideal choice for applications in areas such as signal processing, power systems simulation, and machine learning, where both speed and precision are crucial. However, despite these advantages, CuFP faces challenges when applied to large-scale systems. As the system size and computational complexity increase, resource utilization on the FPGA becomes a critical bottleneck. To address these challenges, we propose an enhancement to the CuFP library by integrating a Self-Alignment Technique (SAT).

8.2.5 Self-Alignment Technique (SAT)

The Self-Alignment Technique (SAT) is an innovative approach designed to optimize floating-point operations, particularly in the context of high-performance hardware implementations. It addresses the critical challenge of single-cycle floating-point accumulation [109]. In Reference [1], a cost-effective method for implementing high-accuracy floating-point operators on FPGAs using the self-alignment technique is presented, enabling the achievement of high clock frequencies.

SAT-Based Addition

SAT-Based Addition represents an approach to optimizing arithmetic operations. Initially introduced to address challenges in single-cycle floating-point accumulation, the SAT has

demonstrated its efficacy in reducing critical path delays in accumulation processes [109,110]. The technique minimizes interactions between the incoming addend and the running sum by aligning the incoming mantissa relative to a common boundary defined by the exponents of the summands. This strategic alignment ensures that fine-grained shifts of the mantissa occur outside the critical path, significantly improving the efficiency of the accumulation loop. Therefore, SAT effectively decouples the computational overhead of mantissa adjustments from the primary addition process, enabling faster and more reliable arithmetic operations.

The main stages of an SAT-based accumulator are detailed in [1]. As illustrated in Figure 8.2, the input `a`, represented as a standard floating-point number, is first unpacked. Unpacking involves decomposing into its constituent parts : sign (s_a), exponent (e_a), and mantissa (m_a). Then, the mantissa is extended (W_{em}) with respect to the ℓ least significant bits of the exponent of `a`. In the next processing stage, accumulation takes place over consecutive cycles. The result in stage 3 must be packed. This stage includes converting to the standard floating-point format after normalization and rounding.

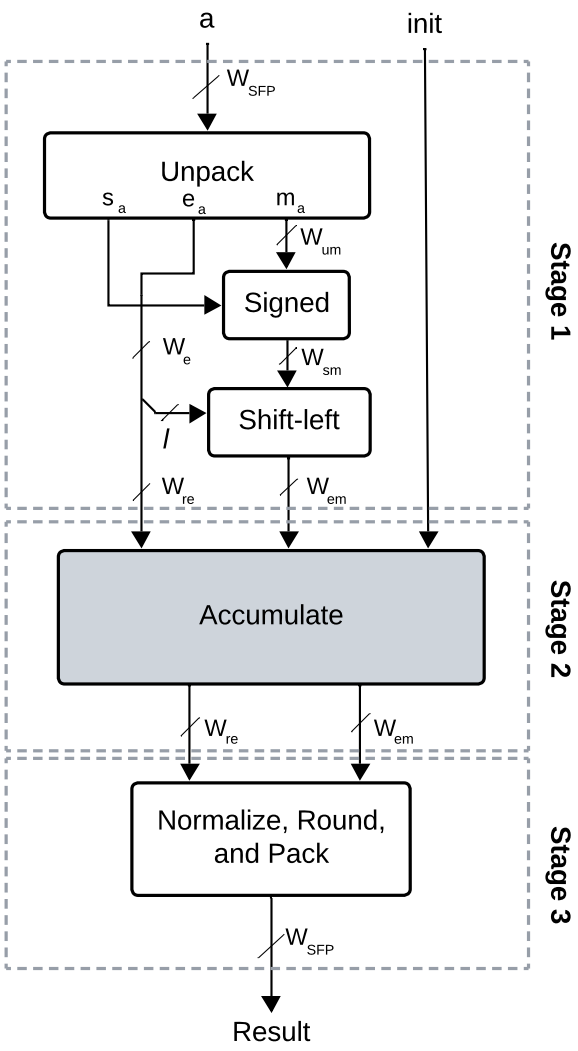


FIGURE 8.2 Main stages of a SAT-based accumulator [1].

Self-Alignment Format (SAF)

The Self-Aligned Format (SAF) [1], which is based on the self-alignment technique [109], is designed to enhance the precision and performance of floating-point operations. The SAF represents a floating-point number x_{fp} that is represented in $SAF(l, w, b)$ as a pair consisting of an integral reduced exponent and an integral extended mantissa, denoted as (e, m). In this notation, l is the number of least significant bits discarded from the standard floating-point exponent ; w is the width of the extended mantissa ; and b is a bias used to adjust the dynamic range. The number x_{fp} is formulated as follows :

$$x_{fp} = m \cdot 2^{2^l(e-b)} \tag{8.2}$$

In this format, the mantissa is first extended by one bit, and the sign is incorporated into the mantissa. Specifically, for negative numbers, the two's complement representation is applied accordingly. Following this, a portion of the exponent, with a length of l bits, is transferred to the mantissa, which extends it by $2^l - 1$ bits. This results in a reduced exponent and an extended mantissa.

Figure 8.3 shows the extended mantissa. To ensure high accuracy during subsequent operations, particularly in the adder tree and accumulator, $\lceil \log_2(N) \rceil$ bits are added to the most significant part of the extended mantissa. Additionally, g guard bits are appended to the least significant part of the mantissa. This process preserves the required precision for the forthcoming computations. In summary, the SAF enhances the mantissa by incorporating part of the exponent and adding extra bits to retain precision. Consequently, the exponent becomes reduced.

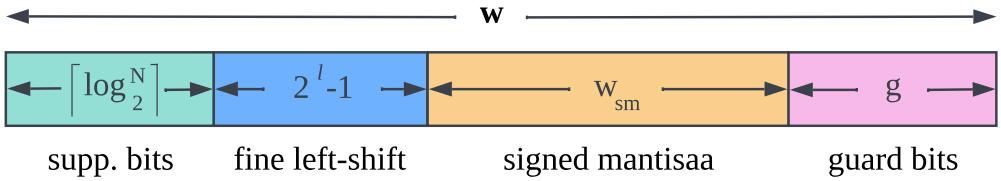


FIGURE 8.3 Extended mantissa composition in SAF [1].

8.3 Implementation Methodology

The SAF format, previously detailed in Section 8.2.5, is seamlessly integrated into the CuFP library to enhance its functionality. The CuFPSAF is implemented using a templated C++ design, allowing for flexibility and scalability by supporting customizable word lengths and exponent sizes. This templated approach ensures that the library can be easily adapted to meet a wide range of application requirements while maintaining high performance and precision. This approach offers several advantages, including allowing users to customize the SAF parameters (`l`, `w`, `b`) based on their specific requirements. Furthermore, integrating SAF into the CuFP library enables users to efficiently handle large-scale systems, leveraging the flexibility of the framework. This adaptability proves particularly valuable for developing algorithms optimized for either high efficiency or low latency, depending on the user's needs.

The choice of coding style plays a critical role in determining the efficiency and performance of the RTL generated by the HLS tool. While object-oriented programming (OOP) in C++ tends to introduce some overhead, it provides significant benefits in terms of code organization, extensibility, and maintainability. In contrast, procedural programming, often seen in

C-style implementations, can be more efficient but might sacrifice readability and flexibility. In this paper, we strike a balance by using a templated class approach and function-based operations. This allows us to retain the efficiency of procedural coding while benefiting from the modularity and clarity offered by OOP. Therefore, the proposed templated class for CuFPSAF enables flexibility in customizing the floating-point format according to the specific needs of the application. This approach allows users to adjust parameters such as word length, exponent size, and bias, making the library adaptable to a wide range of use cases.

8.3.1 CuFPSAF Data Type

The CuFPSAF is a templated class designed to represent and manipulate numbers in the SAF format. It accepts four template parameters :

— `WM` : specifies the bit width of the mantissa.
— `WE` : specifies the bit width of the exponent.
— `l` : indicates the number of bits dropped in the exponent.
— `b` : represents the bias value, calculated based on the selected SAF format.

The CuFPSAF class includes three primary member variables : `m` (mantissa), `e` (exponent), and `lsh` (left shift of original mantissa). These variables collectively define the SAF representation and enable efficient computational operations. Listing 8.1 presents the definition of the CuFPSAF class, omitting the implementation details of the member functions to maintain readability. This modular design facilitates flexible customization and efficient adaptation of the SAF format to specific application requirements.

Listing 8.1 The body of CustomFloat template class.

```
namespace CuFP {
template <int WM, int WE, int l, int b>
class CuFPSAF {
    ap_int<WM> m;     // Mantissa
    ap_uint<WE> e;    // Exponent
    ap_uint<l> lsh;   // Left shift
    ...
};
}
```

8.3.2 Primary Operations

CuFPSAF Addition

The CuFPSAF addition operation leverages the self-alignment technique to efficiently compute the sum of two floating-point numbers, `x` and `y`. Algorithm 3 shows the CuFPSAF addition algorithm. Each floating-point number is defined by its exponent and mantissa, $x \equiv (e_x, m_x, lsh_x)$ and $y \equiv (e_y, m_y, lsh_y)$, respectively. The algorithm begins by determining the input with the larger exponent and storing it as $k \equiv (e_k, m_k, lsh_k)$. Hence, the larger exponent is considered as e_k, and it serves as the base alignment reference (e_z). Using this reference, the alignment shifts for the mantissas are calculated as $q_x = e_k - e_x$ and $q_y = e_k - e_y$. These shift amounts align the mantissas to a common exponent e_z, enabling their addition in the same numerical range. The mantissas are then adjusted and summed as

$$m_z = (m_x \gg (q_x \ll l)) + (m_y \gg (q_y \ll l))$$

where $\gg$ represents a right shift operation to account for the alignment. Finally, the algorithm returns the result $z \equiv (e_z, m_z, lsh_z)$, representing the sum in the CuFPSAF format.

This technique simplifies alignment by introducing the 2^l scaling factor, which reduces hardware complexity. The mantissas are aligned and added efficiently, avoiding costly re-normalization steps.

Algorithm 3 CuFPSAF Addition (inspired from [1])

Parameters : WM, WE, l, b

Inputs : $x \equiv (e_x, m_x, lsh_x)$, $y \equiv (e_y, m_y, lsh_y)$

Outputs : $z \equiv (e_z, m_z, lsh_z)$

Local Variables : $k \equiv (e_k, m_k, lsh_k)$

$k \leftarrow \max(x, y)$ // Compare x and y based on e_x and e_y

$e_z \leftarrow e_k$

$lsh_z \leftarrow lsh_k$

$q_x \leftarrow e_k - e_x$

$q_y \leftarrow e_k - e_y$

$m_z \leftarrow (m_x \gg (q_x \ll l)) + (m_y \gg (q_y \ll l))$

Return z

CuFPSAF Multiplication

The CuFPSAF multiplication operation utilizes the self-alignment technique to efficiently compute the product of two floating-point numbers, `x` and `y`. Algorithm 4 shows the CuFPSAF multiplication algorithm. The algorithm starts by computing the combined exponent e by concatenating the exponents e_x and e_y along with their respective left shifts, then adjusting it by subtracting a constant bias B derived from the exponent width WE and precision width l. This step ensures that the exponents are appropriately scaled for multiplication. The final exponent e_z and left shift lsh_z values are then extracted from the combined exponent e. For the mantissas, the algorithm applies right shifts to m_x and m_y based on their corresponding left shift factors lsh_x and lsh_y. These adjusted mantissas are then multiplied together, and the mantissa result is stored in m_z. Finally, the algorithm returns the result $z \equiv (e_z, m_z, lsh_z)$, which represents the product of x and y in the CuFPSAF format.

Algorithm 4 CuFPSAF Multiplication

Parameters : WM, WE, l, b
Inputs : $x \equiv (e_x, m_x, lsh_x)$, $y \equiv (e_y, m_y, lsh_y)$
Outputs : $z \equiv (e_z, m_z, lsh_z)$
Local Variables : e
Constant : $B \leftarrow 2^{WE+l-1} - 1$
$e \leftarrow concat(e_x, lsh_x) + concat(e_y, lsh_y) - B$;
$e_z \leftarrow e.range(WE + l - 1, l)$
$lsh_z \leftarrow e.range(l - 1, 0)$
$m_z \leftarrow (m_x \gg lsh_x) \times (m_y \gg lsh_y)$
Return z

8.3.3 Vector-Based Operations

CuFPSAF Vector Summation

Vector summation (`vsum`) is an essential operation in computational mathematics and data analysis, where the elements of a vector are aggregated into a single scalar value. For a vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ containing N elements, the summation is expressed as

$$\sum_{i=1}^{N} x_i = x_1 + x_2 + \cdots + x_N \tag{8.3}$$

The templated function `vsum` is designed to accumulate N elements from an array of custom

floating-point numbers represented by the `CuFPSAF` data type.

The `vsum` function implementation is expressed in Algorithm 5, which takes a vector of the `CuFPSAF` object with N elements and returns the result in a scalar CuFPSAF format. The function begins by identifying the index of the element with the maximum exponent in the input array `x[N]` using the `max_index` function.

This function performs a binary search to find the element with the largest exponent, operating with a time complexity of $O(logN)$. As this step is often scheduled to execute in a single clock cycle, it incurs minimal latency. Once the maximum exponent is identified, the corresponding `e` and `lsh` values of the result are directly assigned from the maximum element found. Next, the mantissas of all elements in the input vector are aligned with the maximum exponent by performing a right-shift operation based on their exponent difference. The aligned mantissas are stored in the temporary array `vm`, which is fully partitioned to leverage parallelism. Finally, the summation of all aligned mantissas in `vm` is performed using a tree-based vector addition for integers, implemented as `vsum_int`. This function employs a hierarchical structure that pre-allocates adders for each pair of input elements and iteratively sums intermediate results until a single integer is obtained. With a time complexity of $O(logN)$, `vsum_int` is critical to the overall performance of the `vsum` function, as its latency largely dictates the final latency.

Algorithm 5 CuFPSAF Vector Summation

Parameters : WM, WE, l, b, N

Inputs : $\mathbf{x} = \{x_1, x_2, \ldots, x_N\} \equiv \{(e_{x1}, m_{x1}, lsh_{x1}), (e_{x2}, m_{x2}, lsh_{x2}), \ldots, (e_{xN}, m_{xN}, lsh_{xN})\}$

Outputs : $z \equiv (e_z, m_z, lsh_z)$

Local Variables : $vm[N]$

Constant : $B \leftarrow 2^{WE+l-1} - 1$

$idx_{\max} \leftarrow \text{max\_index}(e_{x1}, e_{x2}, \ldots, e_{xN})$

$k \leftarrow z[idx_{max}]$

$e_z \leftarrow e_k$

$lsh_z \leftarrow lsh_k$

$vm \leftarrow \{m_{x1} \gg ((e_k - e_{x1}) \ll l), m_{x2} \gg ((e_k - e_{x2}) \ll l), \ldots, m_{xN} \gg ((e_k - e_{xN}) \ll l)\}$

$m_z \leftarrow \text{vsum\_int}(vm)$

Return z ;

CuFPSAF Dot Product

The dot product (`dp`) unit is a critical component for numerous computational tasks, particularly in applications involving machine learning, signal processing, and numerical computations. It computes the sum of the products of corresponding elements from two input vectors. Mathematically, for two vectors $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ and $\mathbf{y} = [y_1, y_2, \ldots, y_N]$, the dot product is expressed as

$$DP = \sum_{i=1}^{N} x_i \cdot y_i \tag{8.4}$$

The templated function `dp` implements the dot product operation for two input vectors, `x[N]` and `y[N]`, consisting of custom floating-point numbers represented by the `CuFPSAF` data type. A pseudo-code for this operation is provided in Listing 8.2. The function begins by calculating the element-wise product of corresponding elements from the two input vectors using the `mul` function (line 12). These products are stored in an intermediate vector, which is fully partitioned to eliminate read and write access conflicts and ensure that all operations can proceed without memory bottlenecks (lines 6 and 7). Afterward, the intermediate products are accumulated using the `vsum` function (line 15). Finally, the result is returned as a single `CuFPSAF` object.

Listing 8.2 A pseudo-code of SAFCuFP dot product operation.

```
1  template<int WM, int WE, int l, int b, int N>
2  CuFPSAF<WM, WE, l, b> dp(const CuFPSAF<WM, WE, l, b> x[N],
3                           const CuFPSAF<WM, WE, l, b> y[N])
4  {
5  #pragma HLS INLINE
6    CuFPSAF<WM, WE, l, b> m[N];
7  #pragma HLS ARRAY_PARTITION variable=m type=complete
8
9    for (int i = 0; i < N; i ++) {
10 #pragma HLS UNROLL factor=N
11     // Compute the element-wise products of the corresponding elements
12     m[i] = mul<WM, WE, l, b>(x[i], y[I]);
13   }
14   // Sum of the products
15   return vsum<WM, WE, l, b, N>(m);
16 }
```

This implementation, referred to as the *Latency-Constrained* (LC) approach, offers a signifi-

cant advantage in its ability to process a new input series every clock cycle. This capability greatly enhances throughput and computational power, making it especially well suited for latency-sensitive and real-time applications that demand rapid, responsive computations with minimal delay. However, the main trade-off of this approach is that resource utilization increases linearly with the size of the input vectors, which may pose challenges for hardware with limited resources.

However, many modern applications are designed to process vast amounts of data, often dealing with vectors of large size. These large-scale datasets are increasingly common in fields like machine learning, scientific computing, and big data analytics. In such contexts, operations like the dot product are essential, playing a critical role in tasks such as similarity measurement, optimization, and data transformation. Efficiently processing large vectors is critical for achieving accurate and high-performance results, making the optimization of dot product operations a key area of focus in high-performance computing. Therefore, for very large-scale systems, particularly those with large-sized vectors, the LC approach faces limitations. As the vector size increases, the total resource requirements grow significantly. In FPGA-based applications, this increase in resource usage can become a major issue, as the hardware resources may become overburdened, leading to potential inefficiencies and bottlenecks.

To address these challenges, we propose an alternative approach called the *Resource-Constrained* (RC) approach, which is designed for scenarios where the size of the input vectors is large, and system resources need to be managed efficiently. While this approach is described in the context of the dot product, its methodology can be applied to other computationally intensive operations. A parameter known as the n is introduced to specify how the input vectors are divided into N/n smaller sub-vectors with n elements, enabling sequential processing of partial computations. The results for each sub-vector are progressively accumulated to obtain the final output, as shown in Equation (8.5).

$$DP = \sum_{k=1}^{\frac{N}{n}} \sum_{i=1}^{n} x_i^{(k)} y_i^{(k)} \tag{8.5}$$

This approach strikes a balance between resource utilization and performance by limiting hardware requirements, as it reduces the resources by processing smaller segments of the vector at a time, rather than the entire vector simultaneously. Consequently, it is particularly well suited for scenarios where FPGA resources are limited, and efficient computation of large vectors is essential.

Figure 8.4 compares the two approaches for dot product design. In the LC approach, shown

in Figure 8.4a, the computation follows Equation (8.4), where the entire input vector is processed simultaneously to achieve minimal latency. On the other hand, the RC approach, depicted in Figure 8.4b, divides the input vectors into smaller sub-vectors and allocates the arithmetic logic required for only n input size to reduce hardware resource usage. The partial results coming from the reduced arithmetic unit are progressively accumulated. This approach offers flexibility to address varying design constraints and optimization goals.

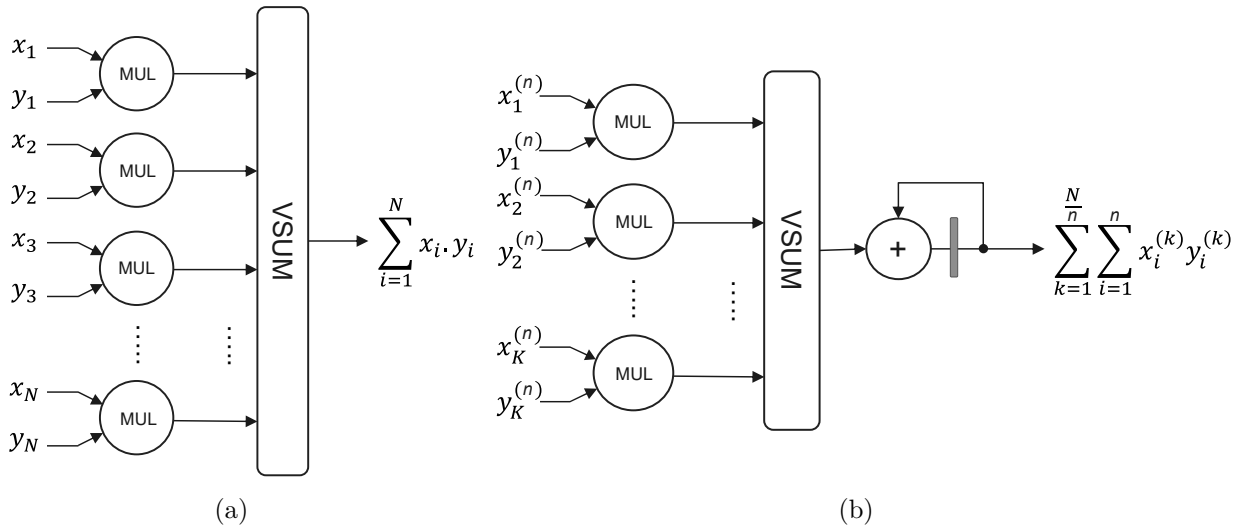


FIGURE 8.4 Dot product design : (a) Latency-Constrained approach ; (b) Resource-Constrained approach. The grey line stands for register level.

Figure 8.5 shows the scheduling of allocated arithmetic units for the dot product operation under the two distinct approaches. In the LC approach (Figure 8.5a), all multiplications (`mul`) are executed in parallel, followed by the vector summation (`vsum`), which minimizes latency at the cost of higher resource usage. Conversely, the RC approach (Figure 8.5b) divides the vector into smaller segments based on a user-defined parameter (n). Each segment undergoes partial dot product computation (`dp`), and the intermediate results are progressively accumulated (`ACC`) over multiple cycles.

For a vector of size N, the LC approach requires N multipliers and a single vector summation unit for size N, with the total latency calculated as

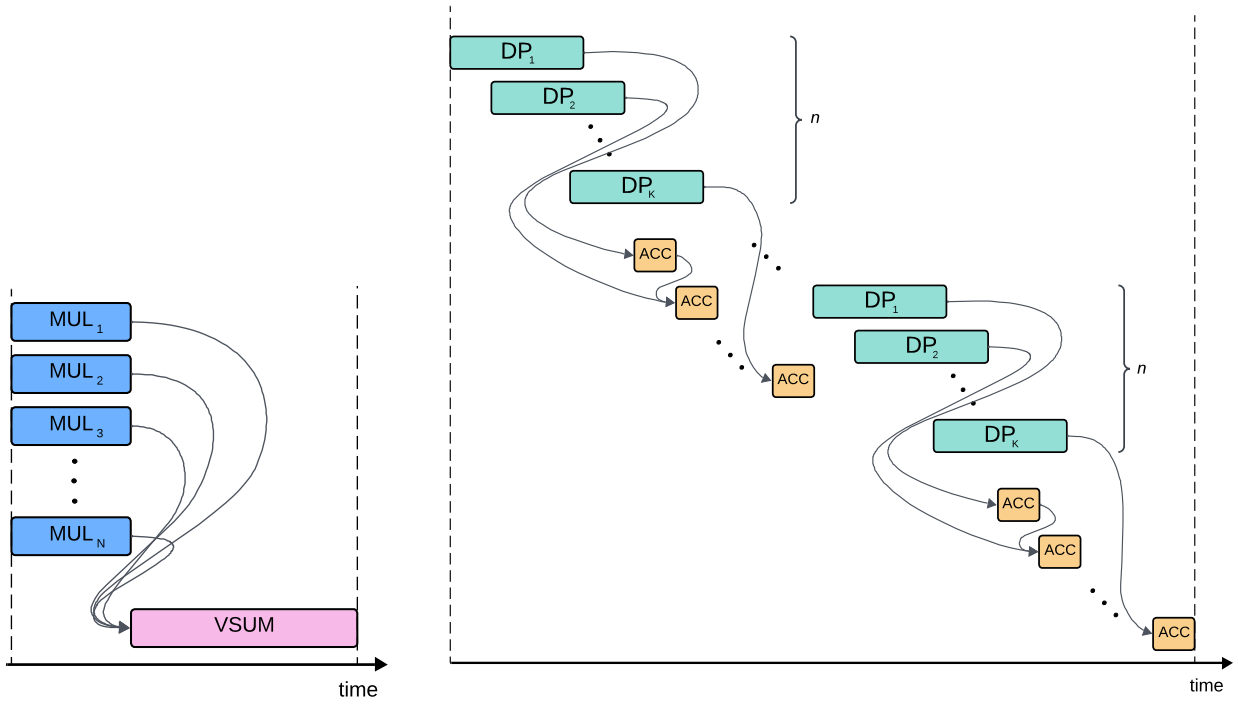
$$\ell_{\mathbf{total}}(\texttt{Latency-Constrained}) = \ell_{\mathbf{mul}} + \ell_{\mathbf{vsum}}$$

In contrast, in the RC approach, the vector is divided into N/n sub-vectors, enabling resource

reuse. The total latency for this approach is given by :

$$\ell_{\mathbf{total}}(\texttt{Resource-Constrained}) = (N/n) + (\ell_{\mathbf{dp}} - 1) + \ell_{\mathbf{ACC}}$$

The following pseudo-code, Listings 8.3, represents managing the dot product operation in both approaches. A wrapper function named `dp_wrapper` is used to handle both approaches using appropriate directives and operations. The choice of approach is determined at compile-time through preprocessor directives (`C_LATENCY` or `C_RESOURCE`). In the LC approach (lines 12–18), the entire input vectors are processed in parallel to achieve minimal latency and maximum throughput. The input vectors are fully partitioned to allow simultaneous access to all elements, and the operation is pipelined with an initiation interval (II) of 1, enabling new inputs to be processed every clock cycle.



(a) (b)

FIGURE 8.5 Schedule viewer : (**a**) Latency-Constrained approach ; (**b**) Resource-Constrained approach. Arrows indicate the data dependencies between operations.

Listing 8.3 A pseudo-code of CuFPSAF dot product operation, considering two different approaches : Latency-Constrained and Resource-Constrained.

```
1 using CuFPSAF_t = CuFPSAF<SAF_WM, SAF_WE, SAF_L, SAF_B>;
```

```
2  #define n 8
3
4  CuFPSAF_t dp_fold(const CuFPSAF_t x[n], const CuFPSAF_t y[n])
5  {
6  #pragma HLS PIPELINE
7    return dp<SAF_WM, SAF_WE, SAF_L, SAF_B, n>(x, y);
8  }
9
10 CuFPSAF_t dp_wrapper(const CuFPSAF_t x[VSIZE], const CuFPSAF_t y[VSIZE
       ])
11 {
12 #if defined(C_LATENCY)
13   // Latency-constrained approach
14 #pragma HLS PIPELINE II=1
15 #pragma HLS ARRAY_PARTITION variable=x type=complete
16 #pragma HLS ARRAY_PARTITION variable=y type=complete
17   return dp<SAF_WM, SAF_WE, SAF_L, SAF_B, VSIZE>(x, y);
18
19 #elif defined(C_RESOURCE)
20   // Resource-constrained approach
21 #pragma HLS PIPELINE
22 #pragma HLS ALLOCATION function instances=dp_fold limit=1
23 #pragma HLS ARRAY_PARTITION variable=x type=complete
24 #pragma HLS ARRAY_PARTITION variable=y type=complete
25
26   CuFPSAF_t sum(0.0);
27   for (int i = 0; i < VSIZE; i += n) {
28 #pragma HLS UNROLL factor=VSIZE/n
29     sum = sum<SAF_WM, SAF_WE, SAF_L, SAF_B>(sum, dp_fold(&x[i], &y[i]))
          ;
30   }
31   return sum;
32 #else
33     #error ``Either C_LATENCY or C_RESOURCE must be defined''
34 #endif
35 }
```

In the RC approach (lines 19–31), the algorithm utilizes two helper functions : `dp_fold` and `sum`. The `dp_fold` function (lines 4–8) acts as a wrapper for the `dp` operation, processing sub-vectors of size `n`. To conserve resources, only one instance of `dp_fold` is allowed, enforced by a directive that restricts instantiation. In `dp_wrapper`, the input vectors are divided into sub-vectors of size `n`, which are processed iteratively. The dot product for each sub-vector is

computed using `dp_fold`, and the results are accumulated in a loop. The loop is pipelined without a fixed initiation interval, allowing the HLS tool to optimize the interval for resource efficiency.

CuFPSAF Matrix-Vector Multiplication

The CuFPSAF matrix-vector multiplication (`mvm`) is a critical component in many high-performance computing applications, particularly those involving linear algebra and machine learning. Conceptually, `mvm` involves computing the dot product between each row of the matrix and the input vector, with each result contributing to an element of the output vector, as follows :

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} \\ x_{21} & x_{22} & \cdots & x_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \cdots & x_{ij} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_j \end{bmatrix} = \begin{bmatrix} x_{11}y_1 + x_{12}y_2 + \cdots + x_{1j}y_j \\ x_{21}y_1 + x_{22}y_2 + \cdots + x_{2j}y_j \\ \vdots \\ x_{i1}y_1 + x_{i2}y_2 + \cdots + x_{ij}y_j \end{bmatrix} \tag{8.6}$$

Since `mvm` requires a large number of additions and multiplications, its direct implementation would demand significant parallel resources, which may not be feasible in FPGA-based designs. To address this, CuFPSAF implements `mvm` using the introduced `dp` computation unit while leveraging the RC approach to balance scalability and performance. The `mvm` function is implemented using a series of `dp` units, where each instance processes a row of the matrix and the vector elements. Listing 8.4 illustrates how the `mvm` function is implemented using the `dp` computation unit. The number of `dp` instances can be adjusted in line 12, where it can take an arbitrary value. Increasing this limit enhances parallelism, reduces latency, but also increases resource consumption, requiring a trade-off between performance and hardware efficiency.

Listing 8.4 A pseudo-code of CuFPSAF matrix-vector multiplication, considering RC approach.

```
1 using CuFPSAF_t = CuFPSAF<SAF_WM, SAF_WE, SAF_L, SAF_B>;
2
3 CuFPSAF_t dp_fold(const CuFPSAF_t x[VSIZE], const CuFPSAF_t y[VSIZE])
4 {
5 #pragma HLS PIPELINE
```

```
6     return dp<SAF_WM, SAF_WE, SAF_L, SAF_B, n>(x, y);
7   }
8
9   void dp_wrapper(const CuFPSAF_t x[VSIZE][VSIZE], const CuFPSAF_t y[
        VSIZE], CuFPSAF_t z[VSIZE])
10  {
11  #pragma HLS PIPELINE
12  #pragma HLS ALLOCATION function instances=dp_fold limit=4
13  #pragma HLS ARRAY_PARTITION variable=x type=complete
14  #pragma HLS ARRAY_PARTITION variable=y type=complete
15  #pragma HLS ARRAY_PARTITION variable=z type=complete
16
17    for (int r = 0; r < VSIZE; r ++) {
18  #pragma HLS UNROLL
19      z[r] = dp_fold(&x[r][0], y);
20    }
21  }
```

8.4 Experimental Results

This section presents the experimental results of implementing the proposed CuFPSAF on the Alveo U280 FPGA using Vitis HLS 2023.2. The target device chosen is the *xcu280-fsvh2892-2L-e*. It should be noted that, for all circuits discussed in this paper, input and output registers are employed to guarantee precise signal handling and readiness for subsequent processing stages. As an example, a pipeline circuit with three levels includes two computational stages, with registers placed at both the input and output for proper synchronization and data flow. Moreover, to ensure a fair comparison across all implementations, we configured the CuFPSAF, CuFP, and Vendor IP to operate within the same numerical precision, specifically in single-precision floating-point format. We set $wm = 64$, $l = 5$ (where $2^l = 32$), and $b = 23 + 127 = 150$ to align with the standard single-precision floating-point representation, where the mantissa consists of 23 bits and the exponent bias is 127. These parameter choices ensure that all implementations are evaluated under equivalent precision constraints, allowing for a meaningful comparison of accuracy, resource utilization, and performance.

Error Evaluation

In this study, we employ the 2-norm relative error (RE_{2-norm}) as the primary metric to quantify the discrepancy between computed and reference outputs. This metric is particularly suited for numerical analysis, as it captures the overall deviation while normalizing the error

relative to the magnitude of the reference output.

The RE_{2-norm} is mathematically expressed as

$$RE_{2-norm} = \frac{\|\text{out}_{\text{ref}} - \text{out}_{\text{com}}\|_2}{\|\text{out}_{\text{ref}}\|_2} \tag{8.7}$$

Here, out_{ref} denotes the reference output, and out_{com} represents the computed output. The *2-norm* ($\|\cdot\|_2$) of a vector is defined as the square root of the sum of the squares of its components, effectively measuring its Euclidean magnitude. The numerator $\|\text{out}_{\text{ref}} - \text{out}_{\text{com}}\|_2$ represents the magnitude of the error between the computed and reference outputs, while the denominator $\|\text{out}_{\text{ref}}\|_2$ scales this error relative to the reference output's magnitude.

This approach ensures that the error metric is dimensionless and normalized, making it suitable for comparing results across different scales and conditions. Smaller RE2-norm values indicate higher accuracy, whereas larger values signify greater deviation from the reference output.

Table 8.1 presents the 2-norm relative error ($RE_{2\text{-}norm}$) evaluation for three floating-point variants : Vendor IP (standard single-precision floating-point), CuFP [4], and CuFPSAF, based on the same set of 100,000 randomly generated numbers. The results focus on `Sum` and `Mul`, as these are the two primary operations reported in CuFP, allowing for a direct and fair comparison. Moreover, `Sum` and `Mul` are fundamental to complex operations like `vsum` and `dp`, and their sensitivity to errors straightforwardly impacts the accuracy of more complex operations. The results show that the errors for the CuFP and CuFPSAF variants are comparable to the Vendor IP, with slightly lower values in most cases.

TABLE 8.1 Error evaluation of CuFP (8, 23), vendor IP (single-precision floating point), and CuFPSAF, based on the same set of 100,000 random numbers.

Operation	**Variant**	$\boldsymbol{RE_{2-norm}}$ (%)
`Sum`	Vendor IP	2.02×10^{-6}
	CuFP [4]	2.68×10^{-6}
	CuFPSAF	2.63×10^{-6}
`Mul`	Vendor IP	6.03×10^{-6}
	CuFP [4]	6.35×10^{-6}
	CuFPSAF	6.31×10^{-6}

The comparison in Table 8.2 highlights the resource utilization and performance of three variants—Vendor IP, CuFP [4], and CuFPSAF—across different vector sizes for the `vsum`

operation. Vendor IP exhibits the highest resource utilization, with a significant number of DSPs, LUTs, and FFs used, but its latency, measured in cycles, is relatively larger. In contrast, CuFP eliminates DSP usage entirely, showing moderate resource usage, though its latency is higher than CuFPSAF. CuFPSAF demonstrates the most efficient performance, achieving the lowest latency and LUT utilization; however, its FF usage surpasses that of CuFP. Notably, CuFPSAF demonstrates a less pronounced increase in FF usage as vector sizes grow, highlighting its scalability and efficiency for larger workloads. This balance between latency, LUTs, and FF scalability underscores CuFPSAF's suitability for lightweight, high-performance designs.

TABLE 8.2 Comparing the resource utilization of `vsum` with different vector sizes for CuFP (8, 23), vendor IP (single-precision floating point), and CuFPSAF at 200 MHz clock frequency.

Variant	**Vector Size**	**Interval**	**# Cycles**	**Resource Utilization**		
				DSP	**LUT**	**FF**
Vendor IP	8	1	12	14	1,564	1,887
	16	1	16	30	3,324	4,007
	32	1	20	62	6,844	8,247
	64	1	24	126	15,364	16,727
CuFP [4]	8	1	5	0	2,010	722
	16	1	6	0	3,662	1,424
	32	1	6	0	7,032	3,891
	64	1	7	0	15,387	6,999
CuFPSAF	8	1	3	0	990	1,197
	16	1	4	0	1,659	2,640
	32	1	4	0	3,644	6,046
	64	1	5	0	7,075	13,757

Table 8.3 presents the resource utilization and performance of the `dp` operation for Vendor IP, Fused Vector FP [50], CuFP [4], and CuFPSAF across various vector sizes. Vendor IP consumes the most resources overall, with a steep increase in DSP, LUT, and FF utilization as the vector size grows. Fused Vector FP, while offering improved latency compared to Vendor IP, requires significantly higher LUT and FF resources, particularly for larger vector sizes,

which may limit its scalability. CuFP, on the other hand, demonstrates moderate resource usage and consistent latency across all vector sizes, making it a predictable and efficient choice. CuFPSAF achieves the shortest latency among all variants and maintains competitive resource utilization, particularly for DSP and LUT usage. However, CuFPSAF's FF utilization increases more significantly than CuFP's, especially for larger vector sizes. This trend highlights the trade-offs involved in achieving low latency and high computational efficiency.

TABLE 8.3 Comparing the resource utilization of `dp` with different vector sizes for CuFP (8, 23), vendor IP (single-precision floating point), and CuFPSAF at 200 MHz clock frequency.

Variant	**Vector Size**	**Interval**	**# Cycles**	**Resource Utilization**		
				DSP	**LUT**	**FF**
Vendor IP	8	1	15	38	2204	3175
	16	1	19	78	4604	6583
	32	1	23	158	10,028	13,399
	64	1	27	318	20,300	27,031
Fused Vector FP [50]	8	1	8	16	6160	2409
	16	1	9	32	12,254	5290
	32	1	13	64	26,094	10,409
	64	1	19	128	51,761	24,977
CuFP [4]	8	1	6	16	2330	1515
	16	1	7	32	4219	2929
	32	1	7	64	8151	5731
	64	1	7	128	16,634	13,796
CuFPSAF	8	1	5	16	3989	2030
	16	1	5	32	8353	3975
	32	1	5	64	16,249	7871
	64	1	5	128	28,208	14,859

Figure 8.6 presents the impact of varying the dividing factor (n) and different vector sizes on resource utilization (LUT, FF, and DSP) and the number of cycles for the CuFPSAF

vector summation operation. For a fixed vector size, increasing n results in reduced latency. For instance, when comparing the results for $n = 4$ to $n = 8$, we observe a reduction in latency across all vector sizes. For smaller vector sizes, such as 8 and 16, the latency remains relatively stable. However, as the vector size increases, the latency reduction becomes more pronounced when moving from $n = 4$ to $n = 8$. This trend continues as we increase n further to 16, with latency continuing to decrease for larger vectors. LUT utilization does not scale inversely with n due to the additional control logic overhead required for managing the RC approach. As n increases, while fewer arithmetic units are needed, the operand scheduling complexity increases, requiring additional control logic. This results in a less pronounced reduction in LUT utilization compared to other resources.

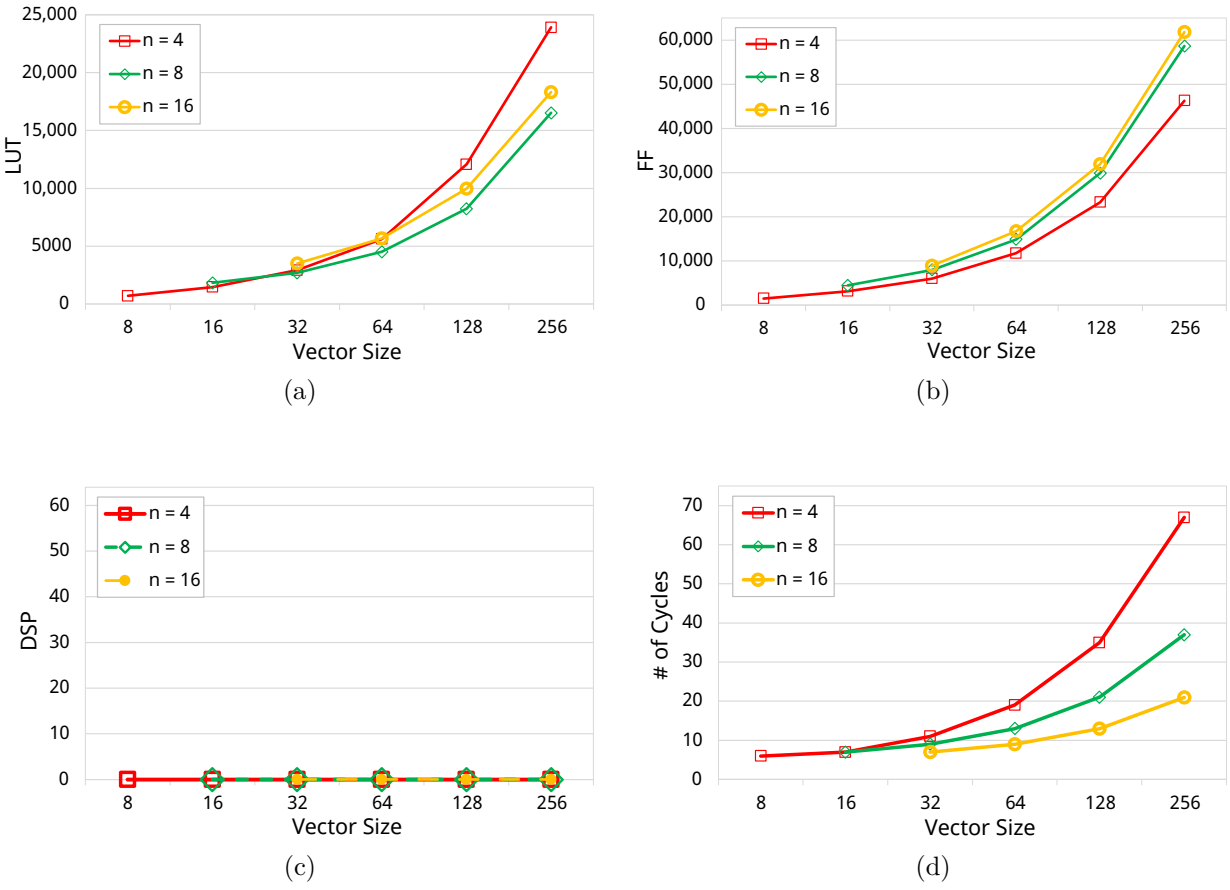


FIGURE 8.6 Comparing the resource utilization of `vsum` operation for different n (4, 8, 16); (a) LUT Utilization; (b) FF Utilization; (c) DSP Utilization; (d) Number of clock cycles.

For different vector sizes, the latency increases as the vector size grows, which is a natural consequence of handling larger data. However, it is important to note that the increase in

latency with vector size is less pronounced for higher n-values. This behavior demonstrates the efficiency of larger dividing factors in managing larger datasets, as they distribute the computational load more evenly and better handle increased workloads. Additionally, for a fixed vector size (N), selecting an appropriate n is crucial; it should neither be too large nor too small.

Figure 8.7 illustrates the impact of varying the dividing factor (n) and different vector sizes on resource utilization (LUT, FF, and DSP) and the number of cycles for the CuFPSAF dot product. As the dividing factor increases ($n = 4$, $n = 8$, $n = 16$), there is a significant rise in resource usage, particularly for LUTs and FFs. This is expected because larger n introduces greater parallelism, which in turn requires more hardware resources. Similarly, DSP usage scales predictably, doubling with each increase in n—from 8 for $n = 4$ to 16 for $n = 8$ and 32 for $n = 16$.

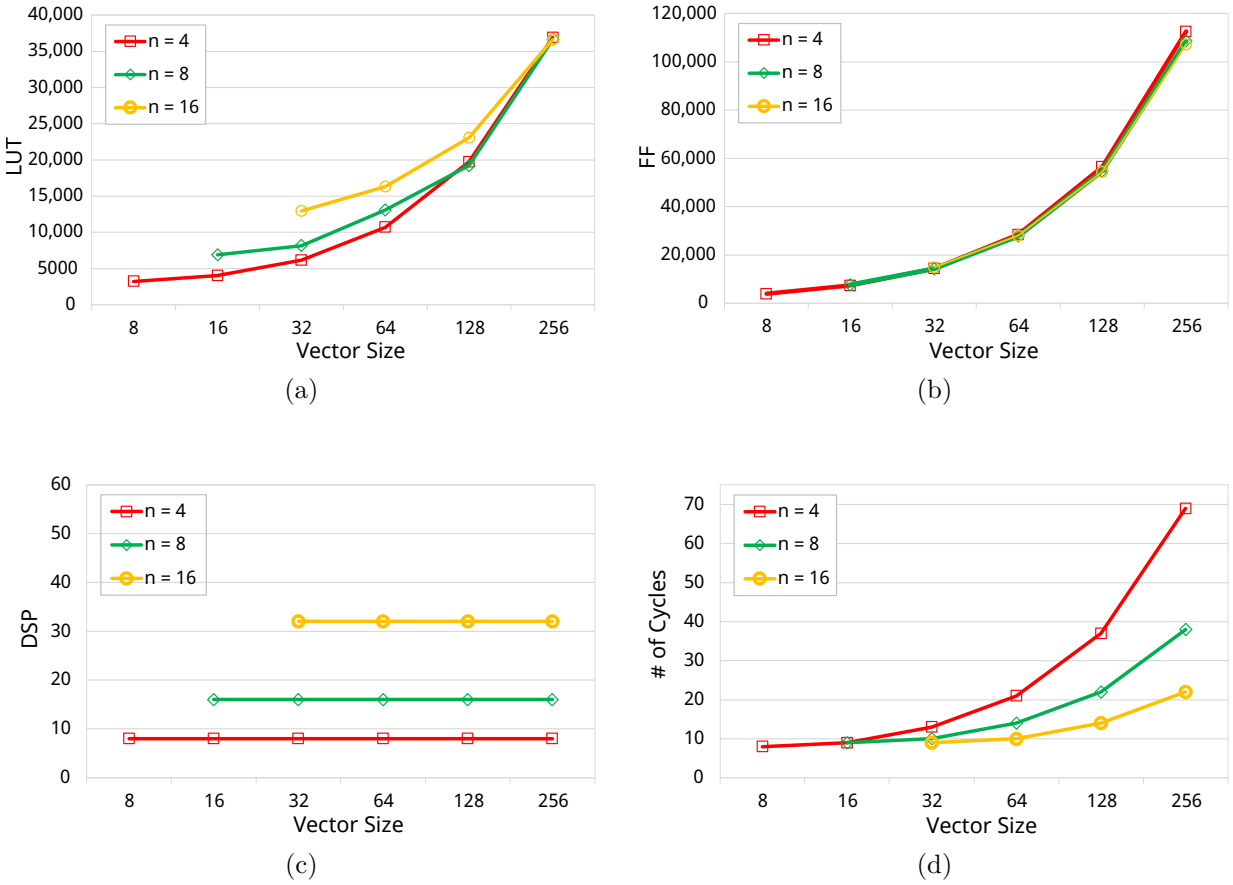


FIGURE 8.7 Comparing the resource utilization of `dp` operation for different n (4, 8, 16); (a) LUT utilization; (b) FF utilization; (c) DSP utilization; (d) number of clock cycles.

For a fixed vector size, increasing n consistently reduces the number of cycles. However, this improvement in latency comes at the expense of higher resource utilization. When considering different vector sizes, a clear trend emerges : the number of cycles increases with larger vectors for the same n. Nevertheless, the increase in latency with vector size is less pronounced for higher n-values, underscoring the efficiency of larger dividing factors in handling more extensive workloads. This behavior indicates that larger dividing factors are better equipped to process larger workloads efficiently as they distribute the computational effort more effectively.

By mitigating the impact of vector size on latency, higher n-values allow for better scalability. For applications with larger vectors, this characteristic becomes particularly advantageous, as it ensures that the latency overhead remains manageable even as the computational demands grow. This highlights the potential of larger dividing factors for optimizing performance in systems designed to handle high data throughput or large-scale processing tasks.

In comparison, the LC approach, as shown in Tables 8.2 and 8.3, achieves minimal latency by performing all computations in parallel. However, this comes at the cost of significantly higher FPGA resource usage, particularly in terms of LUTs and FFs, making it less suitable for resource-constrained applications. The RC approach, as demonstrated in Figures 8.6 and 8.7, processes the small sub-vectors and consequently uses fewer resources, making it more appropriate for scenarios where hardware resources are limited. This trade-off between latency and resource utilization is a critical consideration when selecting the appropriate approach for a given application.

To evaluate CuFPSAF at a larger scale, we compare its performance against CuFP and Vendor IP in matrix-vector multiplication (`mvm`). We consider matrices of sizes 4×4, 8×8, 16×16, and 32×32, with vector sizes of 4, 8, 16, and 32, respectively, meaning that `n` takes values of 4, 8, 16, and 32. Figure 8.8 presents LUT, FF, and DSP utilization, along with latency in clock cycles. LUT consumption, shown in Figure 8.8a, indicates that CuFPSAF requires approximately 100K LUTs, whereas CuFP and Vendor IP exceed 250K and 300K LUTs, respectively, for a 32×32 matrix size. This highlights CuFPSAF's lower LUT footprint.

Figure 8.8b illustrates FF utilization, where the CuFPSAF and CuFP exhibit comparable consumption, but CuFPSAF achieves lower FF usage as matrix dimensions increase. The most significant improvement is seen in DSP utilization, as shown in Figure 8.8c in logarithmic scale. The CuFPSAF requires between 32 and 256 DSPs, whereas CuFP ranges from 32 to 2048 DSPs, and Vendor IP from 72 to 5056 DSPs. This demonstrates that the RC approach in CuFPSAF provides a significantly more resource-efficient solution for large-scale computations. For larger matrix dimensions, maintaining the same performance trends with

CuFP and Vendor IP would become increasingly resource-intensive, necessitating alternative resource management strategies for scalability. In terms of latency, Figure 8.8d shows that CuFPSAF achieves execution times ranging from five to thirteen cycles, whereas CuFP operates between six and seven cycles. While CuFP offers slightly lower latency, this advantage comes at the cost of significantly higher resource consumption, making it unsustainable as computations scale. In contrast, the CuFPSAF balances performance and resource efficiency, making it more suitable for large-scale FPGA-based floating-point operations.

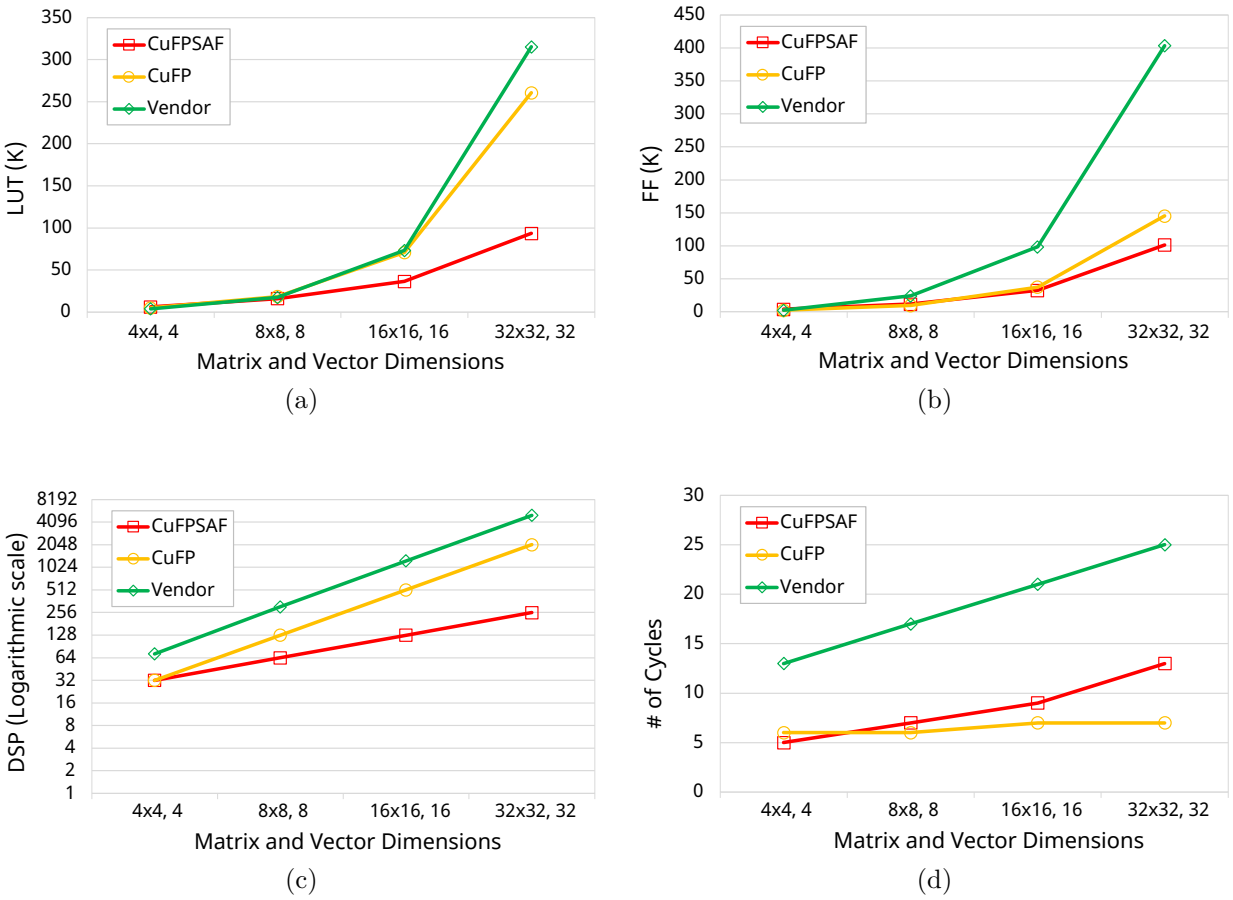


FIGURE 8.8 Comparing the resource utilization of `mvm` operation for different matrix and vector dimensions ; (a) LUT Utilization ; (b) FF Utilization ; (c) DSP Utilization (Logarithmic scale) ; (d) Number of clock cycles. The x-axis represents the matrix and vector dimensions, where, for example, "4×4, 4" denotes a 4×4 matrix multiplied by a 4×1 vector.

8.5 Discussion

By integrating SAF into CuFP in a templated, configurable manner, CuFPSAF significantly

enhances performance and computational efficiency in FPGA-based systems while providing a streamlined HLS framework. This integration also ensures flexibility for varying precision requirements and hardware constraints. The experimental results highlight several key improvements introduced by this approach, including reductions in latency, resource utilization, and error rates, as well as improved scalability across various vector sizes and operational complexities.

The CuFPSAF demonstrates superior performance when compared to both the CuFP library [4], Fused Vector FP [50], and vendor-provided IP cores. Specifically, SAF's ability to streamline alignment operations by reducing the dependency on packing and unpacking stages leads to lower hardware area usage and faster computation times. This is particularly evident in the dot product and vector summation operations, where CuFPSAF achieves the lowest latency and maintains competitive accuracy relative to standard floating-point implementations.

The experimental results also reveal a clear trade-off between latency and resource usage in CuFPSAF designs. For example, while the LC approach achieves minimal latency by executing all operations in parallel, it demands a higher number of hardware resources. In contrast, the RC approach utilizes a dividing factor (n) to process smaller sub-vectors sequentially, which substantially reduces LUT and DSP utilization. However, the increased number of pipeline stages introduced by this method leads to higher FF utilization, as additional registers are inserted to pass the intermediate signals to the next stages. Moreover, the results indicate that the RC method generally requires fewer LUTs compared to the LC method. However, despite using a fixed number of computation units, the LUT usage still increases as N grows. This is due to the need for scheduling and managing sub-vectors of the input data. As N increases, the complexity of the data path also grows, requiring additional control logic to efficiently route and sequence the sub-vectors for computation. This expanded data path demands more LUTs for multiplexing, selection, and scheduling, contributing to the observed increase in resource utilization, as depicted in Figure 8.9. This behavior is related to the tool's handling of the increased complexity and its need to generate additional control logic to manage the growing data path. The trade-off between the LC and RC approaches enables the CuFPSAF framework to cater to a wide range of use cases, from real-time systems requiring high throughput to resource-limited applications where efficiency is paramount.

Another critical observation is the scalability of CuFPSAF for larger vector sizes. By effectively distributing the computational workload, CuFPSAF minimizes the latency impact of increasing data sizes. This scalability is enhanced by the flexibility of the dividing factor, which allows the user to balance performance and resource utilization according to system

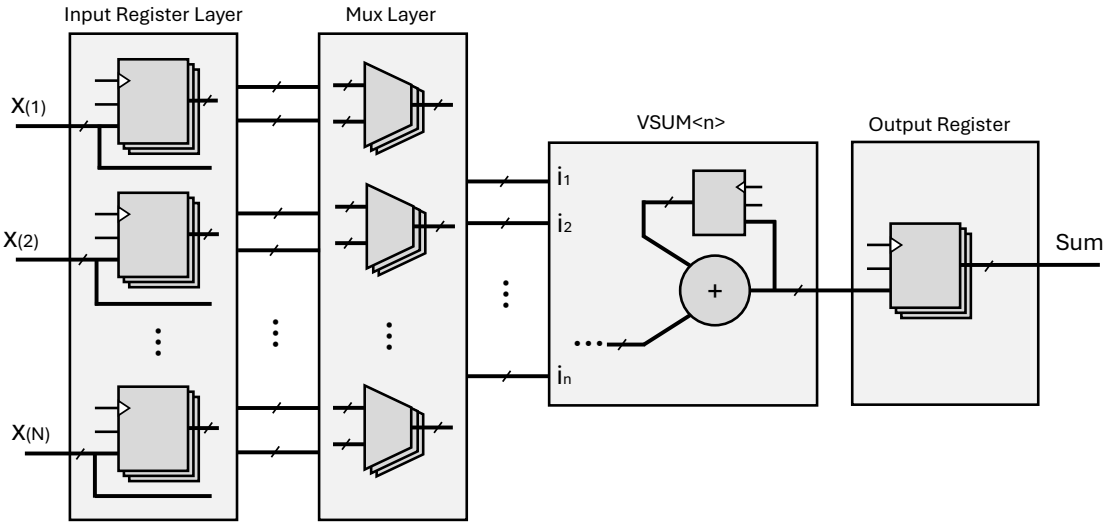


FIGURE 8.9 Input operand scheduling for `vsum` operation with the RC approach.

constraints. The results for large vectors confirm that CuFPSAF can efficiently handle computationally intensive tasks without overburdening the hardware.

Despite these advancements, some challenges remain. The increased usage of FFs in CuFPSAF highlights an area for potential optimization. Future research could explore alternative implementation strategies to mitigate this trade-off while preserving the performance benefits of SAF. Additionally, the CuFPSAF framework could be further extended to support more complex operations, such as matrix multiplications and tensor computations, to broaden its applicability in fields like machine learning and scientific simulations.

8.6 Conclusions

This study demonstrates that integrating the SAF into the CuFP library significantly enhances the efficiency and performance of floating-point operations on FPGA platforms. The CuFPSAF framework achieves notable improvements in latency, hardware resource utilization, and scalability while maintaining high computational accuracy. Specifically, for a vector size of 64, CuFPSAF reduces execution cycles by 29.4% compared to CuFP and by 81.5% compared to vendor IP while utilizing 59.7% fewer DSPs than vendor IP. These advantages make CuFPSAF particularly well suited for high-performance applications, such as real-time simulations, signal processing, and machine learning, where both speed and precision are critical.

The flexibility of the CuFPSAF framework, enabled by its templated design, allows for seam-

less customization to meet diverse application requirements. This adaptability, combined with the ability to balance latency and resource usage through dividing factors, ensures the framework's relevance in a variety of computational scenarios.

Future work will focus on extending the capabilities of the CuFPSAF library to support more complex computational tasks and exploring optimization techniques to further reduce resource usage. Additionally, investigating the integration of CuFPSAF with other high-performance computing platforms could provide valuable insights into its broader applicability.

CHAPITRE 9 ARTICLE 5 : IMPROVED HIGH-RADIX CARRY-SAVE ADDERS FOR LOW-LATENCY VECTOR REDUCTION ON FPGA

Fahimeh Hajizadeh, Paul-André Bisson, Tarek Ould-Bachir, and Jean-Pierre David

Publié dans : 23rd IEEE Interregional NEWCAS Conference (NEWCAS)

Date de soumission : 16 février 2025

Abstract

Efficient summation of multiple inputs is critical in many FPGA-based applications, requiring optimized arithmetic units for reduced latency and resource utilization. This paper explores the high-radix carry-save (HRCS) format and proposes a novel carry chain architecture to improve latency. Moreover, a vector reduction operation using the improved HRCS adders is presented to highlight the impact of the low-latency HRCS adder on high-scale integer additions. Experimental results show that the improved HRCS adder achieves significant delay reductions of up to 29.5% for the `vsum32` in radix-64 format while maintaining nearly the same LUT utilization as the baseline design. Notably, `vsum32` enables summation of up to 32 operands on FPGA in less than 4 ns, showcasing the effectiveness of the proposed HRCS adder in high-speed arithmetic applications with optimized resource efficiency.

9.1 Introduction

Field programmable gate arrays (FPGAs) are reconfigurable integrated circuits that enable high-speed, low-latency data processing, making them ideal for accelerating tasks in signal processing, machine learning, and cryptography [111–113]. Unlike CPUs and GPUs [114], FPGAs enable the creation of custom datapaths tailored to specific applications, offering higher computational efficiency and greater flexibility in optimizing data widths for enhanced performance [115–117]. However, FPGA development faces challenges in efficiently utilizing limited resources, such as logic elements, memory blocks, and routing resources [118]. These constraints are particularly critical for arithmetic circuits, which form the backbone of digital systems. Among these, adders play a crucial role in arithmetic operations, directly impacting the overall performance of FPGA-based designs [119, 120].

Adders are crucial in various applications, ranging from general-purpose processors to specialized hardware accelerators [121, 122]. With the increasing demand for high-performance and

energy-efficient computing, optimizing adder designs for latency and resource utilization has become a significant focus on digital circuit design [123–125]. Traditional adders, such as the ripple carry adder (RCA), offer simplicity and area efficiency but suffer from high latency due to linear carry propagation. Carry look-ahead adders (CLAs) improve latency by parallelizing carry computation at the cost of increased complexity [85], while carry-save adders (CSAs) are particularly advantageous in multi-operand addition scenarios, such as multiplication and accumulation [126]. In [1], a high-radix carry-save (HRCS) adder is introduced, combining the strengths of existing adder architectures while mitigating their limitations. By optimizing carry propagation and reducing critical path delays, the HRCS adder enhances FPGA-based arithmetic operations, improving parallelism and logic utilization. Furthermore, its efficiency in multi-operand computations, such as accumulators, makes it suitable for high-performance computing applications.

In this paper, we build upon the endomorphic HRCS adder introduced in [1] and propose an enhanced version to improve its performance further. Our key contribution lies in refining the carry-chain mechanism to achieve reduced latency while maintaining the crucial endomorphic property. This enhancement optimizes the individual adder's speed. It is also valuable for integration into multi-level adder trees, which is particularly beneficial when connecting multiple instances in series, such as in vector reduction operations. We demonstrate the effectiveness of this approach through extensive experimental results, where vector reduction shows a marked performance improvement, validating the practical benefits of our proposed method. This makes it a valuable and efficient solution for high-performance computing applications, including cryptographic algorithms, real-time signal processing, and machine-learning accelerators.

The remainder of this paper is organized as follows : Section 9.2 provides an overview of the HRCS format and various types of HRCS adders, highlighting their advantages and limitations. Section 9.3 details the proposed improvements to the HRCS adder. Section 9.4 presents the experimental results and analysis, comparing the designs in terms of resource utilization and latency. Finally, Section 9.5 summarizes the contributions and outlines future research directions.

9.2 The High-Radix Carry-Save System

9.2.1 HRCS Format : Definition and Properties

The HRCS format is a redundant number representation system that minimizes carry propagation delays during arithmetic operations. In HRCS, each operand will consist of n digits

A_i ($0 \leq A_i \leq 2^k$), each digit consisting of two components : (i) A sum word $A_i^{(s)}$: this is the k-bit representation of the sum at each digit position ; (ii) A carry bit $A_i^{(c)}$: a single bit representing the carry generated at that position. The HRCS format operates in the high-radix base 2^k, where k is the number of bits used for the sum word. The value of the digit A_i is given by :

$$A_i = A_i^{(s)} + A_i^{(c)} \tag{9.1}$$

HRCS format allows summing multiple digits in parallel without immediately resolving the carry, as shown in Fig. 9.1, with $n = 3$ and $k = 4$. Fig. 9.1 illustrates the addition of `BC7 + 92A = 14F1`. In particular, Fig. 9.1(a) demonstrates the conventional binary + binary ⇒ binary addition, which requires a CRA. Fig. 9.1(b) presents the binary + binary ⇒ HRCS addition, where the $n \times k$ carry chain is decomposed into n independent k-long carry chains, reducing carry propagation delays. Fig. 9.1(c) showcases the binary + HRCS ⇒ HRCS addition, which enables maintaining a running sum in HRCS while incorporating new addends in binary format. Finally, Fig. 9.1(d) explores HRCS + HRCS ⇒ HRCS addition, which is the primary focus of this paper.

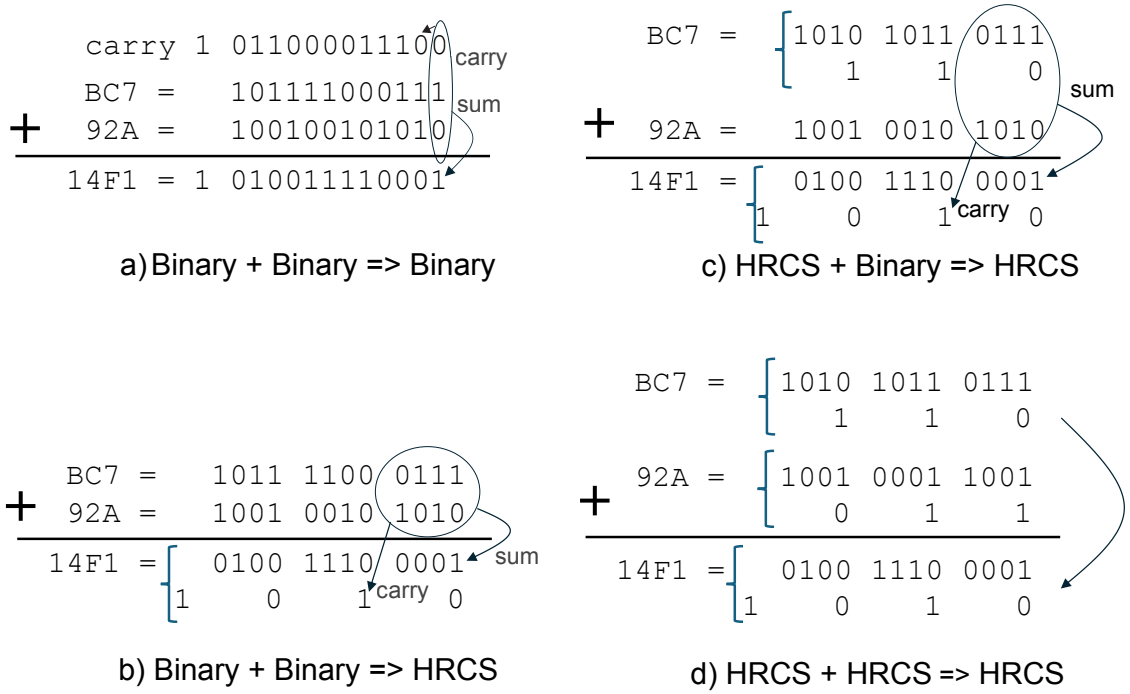
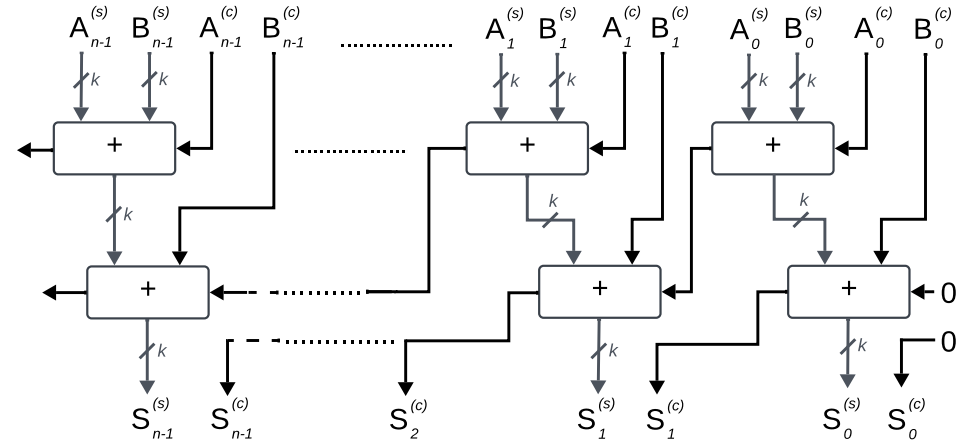


FIGURE 9.1 Illustration of different addition methods involving HRCS representation. (a) Binary + Binary ⇒ Binary addition. (b) Binary + Binary ⇒ HRCS addition. (c) Binary + HRCS ⇒ HRCS addition. (d) HRCS + HRCS ⇒ HRCS addition.

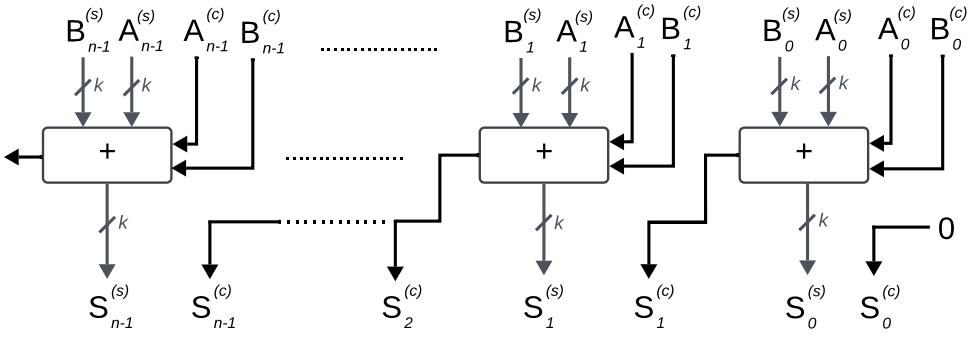
9.2.2 Endomorphic HRCS Adder

An endomorphic HRCS adder — Fig. 9.1(d) — processes two input operands in HRCS format and outputs a result in the same format. This endomorphic property enables the adder to be seamlessly integrated into multi-level adder trees, where the output of one adder can directly serve as the input to the next one without the need for intermediate conversion steps. A key challenge in designing an endomorphic HRCS adder arises from potential overflow when adding two HRCS digits, as their sum can exceed the representational limit of the format [1].

A conventional solution involves using two heterogeneous HRCS adders in sequence, as shown in Fig. 9.2(a) : one for summation and another one for managing the overflow carry. However, such implementation doubles the carry chain length from k to $2k$, significantly increasing the delay and reducing the efficiency. To address this limitation, an endomorphic HRCS adder featuring a single carry-chain has been proposed in [1], as depicted in Fig. 9.2(b).



(a)



(b)

FIGURE 9.2 Endomorphic HRCS adders [1, 2] : (a) Double Carry-chain, (b) Single Carry-chain.

Such architecture is founded on Eq. 9.2 and led to the first FPGA implementation shown in Fig. 9.3.

$$
\begin{aligned}
V_i^j &= A_i^j \oplus B_i^j, && 0 < j < k \\
W_i^{j+1} &= A_i^j \cdot B_i^j, && 0 < j < k-1 \\
\alpha_{i+1} &= A_i^j \cdot B_i^j, && j = k-1 \\
\beta_i &= A_i^0 \oplus B_i^0 \oplus A_i^{(c)} \oplus B_i^{(c)} \\
\gamma_i &= A_i^0 \cdot B_i^0 \cdot A_i^{(c)} \cdot B_i^{(c)} \\
W_i^{(c)} &= \alpha_i \cdot \beta_i + \gamma_i \\
W_i^1 &= (A_i^0 + B_i^0 + A_i^{(c)})(A_i^0 + B_i^0 + B_i^{(c)}) \\
&\quad (A_i^0 + A_i^{(c)} + B_i^{(c)})(B_i^0 + A_i^{(c)} + B_i^{(c)}) \\
S_i^0 &= \alpha_i \oplus \beta_i \\
\alpha_0 &= 0
\end{aligned}
\tag{9.2}
$$

The adder depicted in Fig. 9.3 operates on two 2^4-radix HRCS operands, A and B. As described in Eq. 9.2, V_i and W_i represent the partial sum words, accompanied by the α_{i+1} bits at the carry position. A carry propagation step is performed on V_i, W_i^s, and W_i^c, generating a partial sum word $\hat{S}_i^s = S_i^{k-1} \dots S_i^1$ and a carry bit S_{i+1}^c. The final sum word is obtained by concatenating $S_i^s = \hat{S}_i^s S_i^0$. Fig. 9.3 illustrates the implementation for a 2^4-radix adder, but this procedure can be generalized for higher values of k, as FPGA slices are inherently organized in columns, allowing scalable extensions.

This endomorphic HRCS adder with single carry-chain provides notable benefits, including reduced latency and seamless integration into multi-level adder trees. However, further optimizations to improve its efficiency are possible. This work introduces an enhanced endomorphic HRCS adder featuring a novel carry handling mechanism designed to minimize critical path delay, resulting in lower latency.

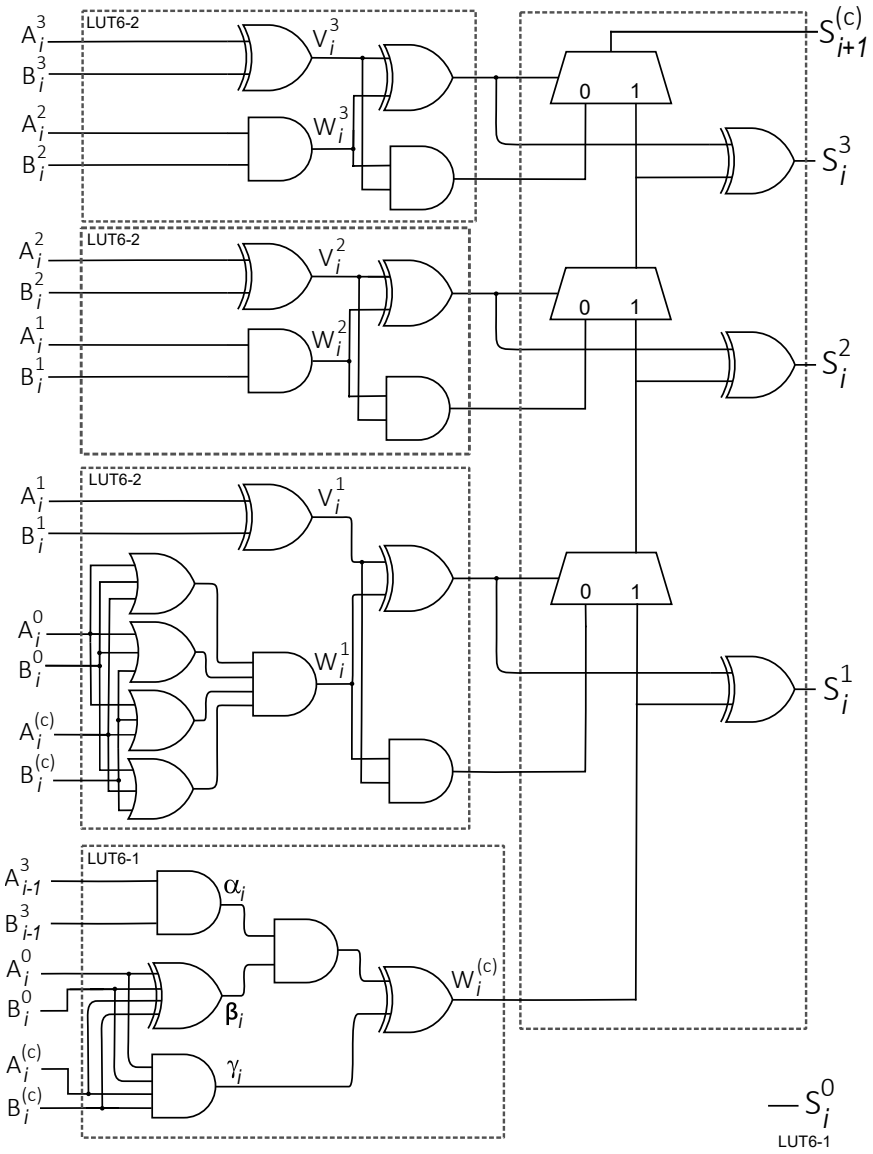


FIGURE 9.3 Endomorphic 4-radix HRCS adder (single carry-chain) implementation [2].

9.3 Improved Endomorphic HRCS Adder

9.3.1 Internal architecture of the Endomorphic HRCS Adder

The proposed architecture for our improved endomorphic HRCS adder is illustrated in Figure 9.4. As already the case in the original architecture (Fig 9.3), V_i^j represents the sum bit, while W_i^j and $W_i^{(c)}$ manage the intermediate carry propagation without the need for full carry resolution. The only difference is in the handling of the carry chain. Namely, the AND gate taking V_i^j and W_i^j as inputs has been removed because it does not impact the

carry computation. Actually, when both signals are different, the multiplexer does not use the output of the AND gate, and when both signals are the same, the AND gate is useless. This change does not have a real impact on the LUT6-2 delay since all the combinations of the inputs are pre-computed inside a Look-up table.

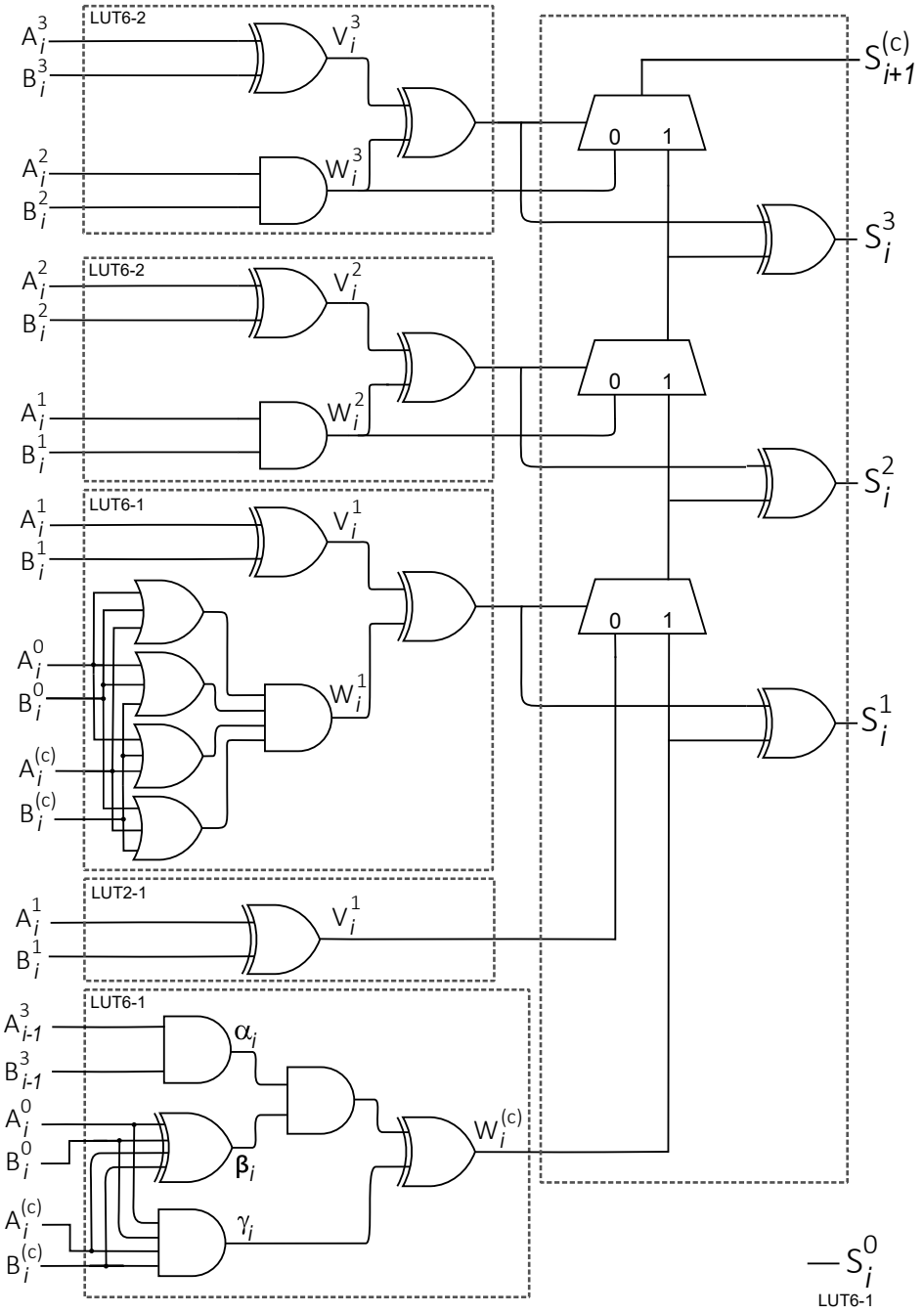


FIGURE 9.4 Improved endomorphic 4-radix HRCS adder (single carry-chain) implementation.

Nevertheless, the impact comes from the very beginning of the carry chain, which better fits the internal architecture of the internal carry chain inside the FPGA. Namely, in order to force the synthesis tool to minimize the delay of the 3-bit operands addition $V_i + W_i$ with an input carry $W_i^{(c)}$, we use the following VHDL statement :

```
sum <= (v & wCin(i) & '1') + (w & '0' & '1');
```

The two LSB result bits are finally ignored, but the carry $W_i^{(c)}$ signal (`Cin` in the VHDL statement) is correctly propagated in the carry chain of the FPGA. This approach enables parallel carry propagation, reducing delay and redundant logic.

A key feature of the improved HRCS adder is its ability to maintain a short critical path compared to original implementations. This is achieved by associating the initial carry-in with a dedicated LUT that is tightly integrated into the carry chain. This design approach forces the synthesis tool to place the carry-in LUT in close proximity to the modules that require it, thereby preventing unnecessary routing delays. In the conventional design, the synthesis process does not always guarantee optimal placement of the carry-in LUT, which can lead to increased critical path delays.

9.3.2 Vectorized Endomorphic HRCS Adder

To further demonstrate the advantages of the original and improved HRCS adders, we implemented vector reduction, a computationally intensive operation that highlights the importance of low-latency and high-performance arithmetic structures. Vector reduction adds all the components of a vector in a single cycle, making the delay of individual adders a critical factor in overall performance. This is particularly relevant in tree-based summation architectures, where multiple adders are placed in series, and even negligible delays in each adder can accumulate, significantly impacting the total computation time.

9.4 Experimental Results

This section presents the experimental results of implementing various types of HRCS adders and evaluates their performance in the context of parallel adder designs. The implementations were carried out on the Alveo U280 FPGA platform using Vivado 2023.2, with the target device set to the *(xcu280-fsvh2892-2L-e)*[1]. It should be noted that for all circuits discussed in this paper, input and output registers are employed to ensure precise signal handling, with the reported delays accurately reflecting the true critical path delays. The primary objective

1. The source code for this work is publicly available as open-source in the following repository : https://github.com/FahimeHajizadeh/HRCS

of these experiments was to analyze the resource utilization, latency, and efficiency of the proposed adder architectures.

Table 9.1 presents a comparison of LUT utilization and delay for three HRCS adder architectures : Endomorphic HRCS with a double carry-chain (EDC-HRCS), Endomorphic HRCS with a single carry-chain (ESC-HRCS) [1], and Improved Endomorphic HRCS with a single carry-chain (IESC-HRCS). The evaluation is conducted across different values of k (bits per HRCS digit). As expected, both LUT usage and delay increase with k for all architectures. The EDC-HRCS exhibits the highest LUT consumption and delay due to the use of two successive HRCS adders, leading to increased resource usage and longer propagation times. In contrast, the ESC-HRCS architecture significantly reduces both LUT utilization and delay by utilizing a single carry-chain. The IESC-HRCS further improves performance, achieving the lowest delay among the three designs due to its novel carry-chain mechanism, which enhances efficiency by minimizing processing stages.

TABLE 9.1 LUT and delay comparison for different HRCS adder architectures across different HRCS digit (k).

Adder Type	**Metric**	**Bit per HRCS digit (k)**					
		2	**4**	**8**	**16**	**32**	**64**
EDC-HRCS	LUT	49	127	256	512	1024	2047
	Delay (ns)	0.87	1.29	1.29	1.30	1.41	1.62
ESC-HRCS [1]	LUT	63	111	207	288	544	1056
	Delay (ns)	0.84	1.08	1.10	1.13	1.18	1.29
IESC-HRCS	LUT	63	110	207	288	544	1056
	Delay (ns)	0.84	0.92	0.92	0.93	1.01	1.11

The vectorized reduction of 8 (`vsum8`), 16 (`vsum16`), and 32 (`vsum32`) inputs were implemented with N = 16 using different k-radix values. The implementation and evaluation were conducted exclusively on the vector reduction constructed by the ESC-HRCS and IESC-HRCS, while the EDC-HRCS adder was excluded due to its inefficiencies in both delay and logic occupation. The delay and occupied LUT for vsum8, vsum16, and vsum32 are shown in Figs. 9.5 and 9.6, respectively. As k increases, both architectures exhibit rising LUT consumption and delay. The IESC-HRCS `vsum` offers significant delay reduction while maintaining nearly the same LUT usage as the ESC-HRCS `vsum`. The main advantage is in delay reduction, which becomes more pronounced as the radix increases. For radix 16 and 32, the IESC-HRCS `vsum8` reduces delay by 19.0% and 19.6%, respectively. The delay improvement

continues to scale, reaching 24.3% at radix 64. This reduction reaches 28.3% and 29.5% for `vsum16` and `vsum16` at radix 64, respectively. This demonstrates that the IESC-HRCS `vsum` achieves better timing efficiency, especially for larger radix values, without additional resource overhead.

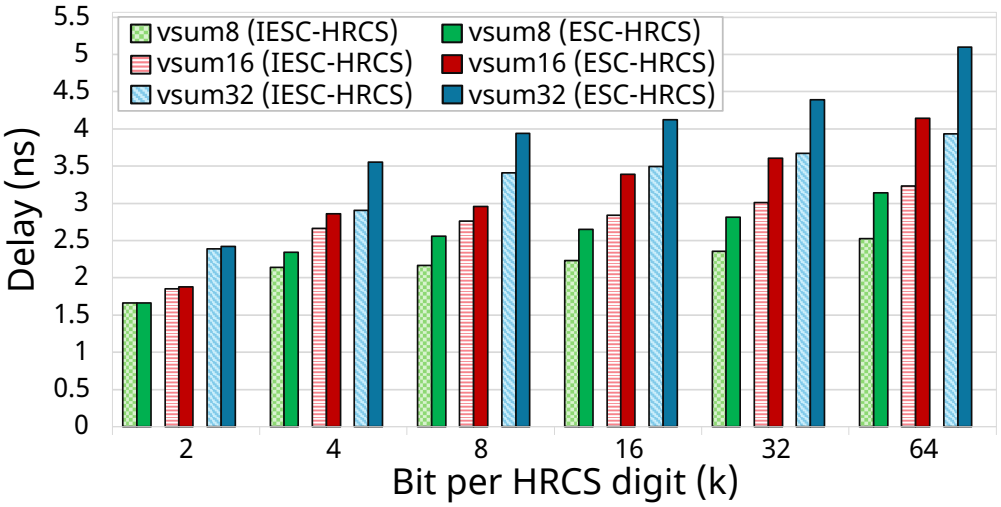


FIGURE 9.5 Delay comparison for the `vsum` across different HRCS digit (k).

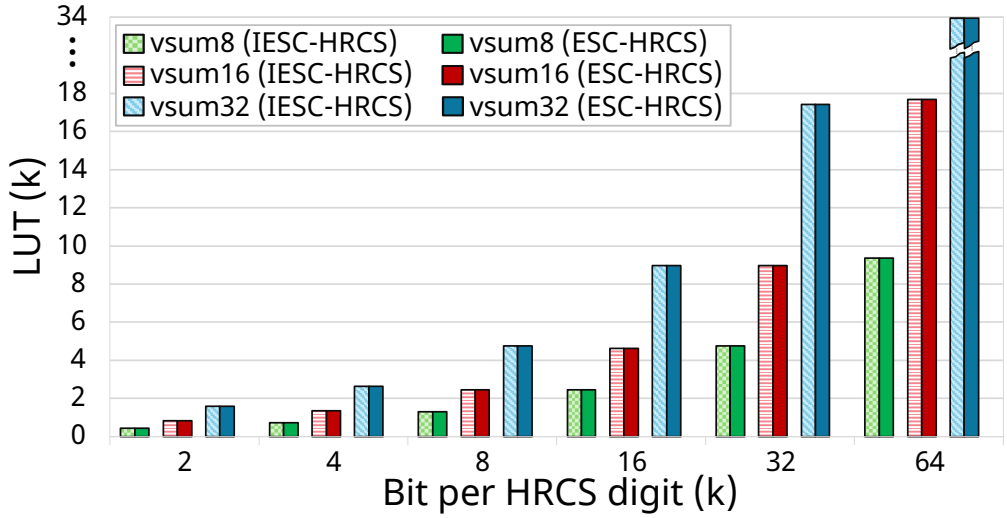


FIGURE 9.6 LUT comparison for the `vsum` across different HRCS digit (k).

9.5 Conclusion

This paper presented an improved endomorphic HRCS adder that enhances carry propagation efficiency while maintaining the HRCS format. By integrating a multiplexer-based carry

chain structure, the proposed design ensures seamless carry resolution and reduces critical path delays. Compared to conventional HRCS adders, the improved design demonstrates superior scalability, making it a strong candidate for multi-level adder trees. Hence, a vectorized summation operation based on the improved HRCS adder is introduced to target high-performance applications. The architecture is particularly well-suited for FPGA-based implementations, where latency is crucial. The 32-input vectorized sum gains 29.5% lower latency at radix-64 format compared to the vectorized summation with the normal single-chain HRCS adder. Future work will focus on integration into arithmetic processing units and extension to complex operations like dot product arithmetic and matrix multiplication.

CHAPITRE 10 DISCUSSION GÉNÉRALE

10.1 Rappel de la problématique

L'utilisation de simulateurs en temps réel joue un rôle fondamental dans le test et le développement des réseaux électriques modernes, en particulier à mesure que la complexité et les exigences de performance dynamique de ces systèmes continuent d'augmenter. Cependant, la conception et l'implémentation de ces simulateurs, notamment ceux reposant sur des plateformes FPGA, demeurent des tâches techniquement complexes.

Dans ce contexte, l'objectif de cette recherche était de proposer une méthodologie visant à faciliter l'implémentation de simulateurs en temps réel sur FPGA, afin de les rendre plus accessibles aux spécialistes applicatifs et aux ingénieurs en commande, ne disposant pas nécessairement d'une expertise approfondie en conception matérielle.

10.2 Contributions de la thèse à la problématqiue

Les contributions de cette thèse s'articulent autour d'un enchaînement méthodologique cohérent, divisé en trois étapes principales.

Dans un premier temps, une approche de modélisation basée sur la représentation pôles-résidus des FDNE a été introduite. Cette formulation a été implémentée à l'aide de la synthèse de haut niveau pour démontrer la faisabilité de transposer des modèles dynamiques complexes vers des architectures FPGA. Dans le cadre de cette première étape, une analyse approfondie des différentes résolutions numériques a été menée. Les formats à virgule flottante standard (simple précision et double précision), ainsi que l'arithmétique en virgule fixe, ont été évalués en termes de latence, d'utilisation des ressources et de précision numérique. Les résultats ont montré que, si le format double précision offre une précision supérieure, il entraîne également un coût important en termes de logique et de latence. À l'inverse, les formats en virgule fixe permettent une implémentation plus efficace sur le plan matériel, au prix d'une réduction de la plage dynamique. Ce compromis a mis en évidence la nécessité d'une représentation numérique plus flexible, pouvant être adaptée aux contraintes et aux exigences de précision des simulations en temps réel sur FPGA.

En réponse à cette problématique, la deuxième étape de ce travail a introduit une nouvelle bibliothèque HLS pour l'arithmétique en virgule flottante personnalisée, permettant un contrôle fin des largeurs de l'exposant et de la mantisse. Cette bibliothèque a été conçue pour combi-

ner les avantages de la large plage dynamique des formats flottants avec l'efficacité matérielle propre aux représentations en virgule fixe. L'intégration d'un format auto-aligné a permis d'optimiser davantage la chaîne de traitement des additions, réduisant ainsi la latence et la consommation de ressources. Ces développements ont abouti à une infrastructure modulaire et portable, adaptée à la conception d'unités arithmétiques efficaces pour les applications en temps réel. Enfin, la troisième étape s'est concentrée sur l'application des stratégies arithmétiques proposées à des cas de simulation plus complexes. Deux approches distinctes de modélisation en espace d'état des FDNE ont été proposées et évaluées, en mettant l'accent sur leur compatibilité avec la simulation en temps réel sur FPGA. Ces techniques de modélisation ont ensuite été appliquées à la simulation en temps réel d'un STATCOM connecté au réseau, en utilisant les bibliothèques HLS développées au cours des étapes précédentes.

10.3 Cohésion des publications

Les cinq articles présentés dans cette thèse s'inscrivent dans une démarche de recherche structurée et progressive, articulée autour du défi central de la simulation en temps réel sur FPGA. Chacun de ces travaux aborde un niveau spécifique de la chaîne de conception, allant de la modélisation des réseaux électriques jusqu'à l'implémentation matérielle d'opérateurs arithmétiques optimisés, en passant par la définition de représentations numériques adaptées. L'enchaînement des contributions témoigne d'une méthodologie cohérente, chaque développement s'appuyant sur les acquis des précédents. Ensemble, ces articles constituent un ensemble cohérent et solide de travaux répondant directement à la problématique de recherche initiale. Un résumé de chaque article est présenté dans les sous-sections suivantes.

10.3.1 Article 1

Dans le chapitre 5, nous avons proposé une méthode de simulation en temps réel des systèmes électriques dans les avions modernes en utilisant des FPGAs, dans le but de surmonter les limites des méthodes de test traditionnelles et des approches de modélisation simplifiées. L'accent a été mis sur la modélisation précise des câbles courts des avions à l'aide de modèles FDNE, qui permettent de capturer des effets d'impédance complexes souvent négligés dans les circuits RL de base. Une approche basée sur la décomposition pôles-résidus a été adoptée pour la modélisation FDNE et mise en œuvre à l'aide de la synthèse de HLS, avec une intégration selon Euler implicite et une interaction avec un solveur MANA. L'objectif principal était d'atteindre des pas de temps de simulation inférieurs à la microseconde. Dans ce but, nous avons étudié en détail l'impact de la résolution numérique et appliqué diverses optimisations HLS, telles que les pragmas, pour améliorer les performances.

Des pas de temps inférieurs à la microseconde ont été obtenus pour tous les types de données mais cette performance a été démontrée sur un cas test de petite taille uniquement. Comme prévu, le pas de temps augmente avec la taille du réseau. Parmi les différents formats numériques, la représentation en virgule fixe a permis de réduire significativement la latence et l'utilisation des ressources par rapport aux formats en virgule flottante simple et double précision, tout en maintenant un compromis acceptable en matière de précision. Par exemple, dans un modèle à 2 ports et 16 pôles, le format à virgule fixe 9.18 a atteint une latence de 80 ns, tandis que les formats à virgule flottante simple et double précision nécessitaient respectivement 245 ns et 255 ns. En termes d'utilisation des ressources DSP, le format en virgule fixe a utilisé moins de 5 % des ressources, contre 14,13 % pour la simple précision et 30,83 % pour la double précision.

Ces résultats montrent que le format en virgule fixe constitue une solution rapide, économe en ressources et suffisamment précise pour la simulation en temps réel des réseaux électriques aéronautiques. Toutefois, les limites en termes de plage dynamique demeurent un défi majeur pour ce type de représentation. Il apparaît donc nécessaire de poursuivre les recherches afin de combiner les avantages respectifs des formats en virgule flottante et en virgule fixe. Cette étude constitue notre première contribution dans ce domaine.

10.3.2 Article 2

Dans le Chapitre 6, nous nous sommes concentrés sur la question de la résolution numérique, en mettant particulièrement l'accent sur les formats personnalisés. En réponse à la flexibilité limitée offerte par les cœurs IP à virgule flottante standard disponibles dans les outils HLS commerciaux — lesquels ne prennent généralement en charge que des formats IEEE-754 prédéfinis tels que les formats demi, simple et double précision — nous avons proposé une nouvelle bibliothèque de synthèse de haut niveau nommée CuFP. Le développement de CuFP a été motivé par le besoin de mieux contrôler la représentation numérique, notamment dans des applications telles que la simulation HIL, où la réduction de la latence et l'optimisation de l'utilisation des ressources sont essentielles. Contrairement aux solutions proposées par les fournisseurs, qui imposent une précision fixe et manquent de configurabilité, CuFP permet aux utilisateurs de définir, à la compilation, les tailles de mot de l'exposant et de la mantisse. Cette fonctionnalité permet la prise en charge de calculs en précision mixte et offre des compromis précis entre plage dynamique, précision et efficacité matérielle. La bibliothèque CuFP proposée inclut des implémentations optimisées des opérations arithmétiques fondamentales couramment utilisées dans les calculs scientifiques et techniques, telles que la sommation vectorielle (`VSUM`), le produit scalaire (`DP`) et la multiplication matrice-vecteur (`MVM`). L'im-

plémentation repose sur une architecture C++ basée sur des modèles (templated), combinée à des directives d'optimisation HLS et à un processus automatisé de génération de cœurs IP.

Les évaluations expérimentales ont montré que CuFP surpasse les blocs IP des fournisseurs en termes de latence et d'utilisation des ressources, notamment pour les opérations vectorielles et matricielles. Par exemple, pour une multiplication matrice-vecteur impliquant une matrice 32×32 et un vecteur de 32 éléments, CuFP atteint une latence de seulement 7 cycles d'horloge, contre 22 cycles pour les IP fournisseurs, tout en consommant environ 60 % de DSP en moins, 10 % de LUT en moins, et 60 % de FF en moins. De plus, pour les opérations de base telles que l'addition et la multiplication, CuFP atteint des fréquences d'horloge maximales plus élevées, tout en affichant une utilisation des ressources comparable, voire meilleure, que les solutions alternatives telles que FloPoCo ou les cœurs propriétaires.

10.3.3 Article 3

L'intégration croissante des convertisseurs électroniques de puissance — tels que ceux utilisés dans les systèmes éoliens, les installations photovoltaïques et les véhicules électriques — dans les réseaux électriques modernes introduit des défis importants en matière de stabilité du réseau et de performance dynamique. Tester ces stratégies de manière précise et efficace est essentiel, mais peut s'avérer très exigeant sur le plan computationnel, en particulier pour les simulations de transitoires électromagnétiques.

Pour répondre à cette problématique, le Chapitre 7 a présenté un cadre de simulation en temps réel basé sur FPGA, spécifiquement conçu pour les convertisseurs raccordés au réseau. Ce cadre intègre un modèle de type FDNE permettant de représenter de manière efficace les dynamiques complexes et dépendantes de la fréquence du réseau externe, simplifiant ainsi le système à étudier tout en préservant une grande précision.

Le chapitre a fourni une analyse détaillée de l'intégration des principaux composants de la simulation à l'aide d'une formulation en espace d'états. Deux stratégies d'intégration distinctes ont été proposées et évaluées, en mettant particulièrement l'accent sur leur impact respectif sur la latence de calcul. Un STATCOM connecté à un réseau de transport haute tension a été utilisé comme cas test représentatif afin de valider le cadre de simulation proposé. La précision de la simulation a été évaluée par comparaison avec un modèle de référence développé sous EMTP®. Les résultats ont montré une excellente concordance entre tous les formats numériques testés, confirmant la robustesse et la fiabilité de l'approche proposée. Un aspect clé de cette étude a été l'intégration des représentations numériques précédemment développées, en particulier la bibliothèque CuFP. L'évaluation de la précision numérique a été basée sur l'erreur relative en norme 2 (RE 2-norm), avec des valeurs observées allant

d'environ 0,0103 % pour la double précision et CuFP (11, 53) à 0,0868 % pour CuFP (8, 24). Ces résultats confirment la grande fidélité computationnelle atteinte par le cadre proposé.

L'implémentation sur FPGA a permis d'atteindre des latences d'exécution inférieures à la microseconde. Parmi les deux approches évaluées, la seconde a systématiquement permis de réduire la latence, au prix toutefois d'une consommation accrue des ressources matérielles. Par exemple, en utilisant la représentation flottante en simple précision, l'approche 1 a atteint une latence de 580 ns, tandis que l'approche 2 l'a réduite à 492 ns. De même, en double précision, la latence est passée de 720 ns avec l'approche 1 à 628 ns avec l'approche 2. Par ailleurs, le cadre proposé a démontré une utilisation efficace des ressources matérielles sur la plateforme FPGA. En sélectionnant des configurations CuFP appropriées, les utilisateurs ont pu naviguer entre les compromis liés à la précision numérique et à la consommation des ressources matérielles, renforçant ainsi la flexibilité et l'applicabilité du cadre de simulation proposé pour une large gamme d'applications en temps réel dans les systèmes électriques.

D'autre part, pour les systèmes de grande envergure ou les configurations à haute précision, les capacités de personnalisation offertes par CuFP peuvent entraîner une augmentation des délais sur le chemin critique et nuire aux performances, en particulier pour les opérations complexes telles que le produit scalaire. Dans de nombreux cas, les optimisations traditionnelles au niveau de la synthèse haut niveau se sont révélées insuffisantes pour atténuer efficacement ces limitations.

10.3.4 Article 4

Pour remédier à cette contrainte, le chapitre 8 a présenté une version améliorée de la bibliothèque CuFP, nommée CuFPSAF, qui intègre la Self-Alignment Technique (SAT) ainsi que son format associé, le Self-Aligned Format (SAF). Le format SAF est spécifiquement conçu pour améliorer l'efficacité de l'alignement des exposants et des mantisses dans l'arithmétique en virgule flottante, en déplaçant les opérations de mise en paquet/dépaquetage et les décalages fins de la mantisse hors du chemin critique des données. Cela permet de réduire de manière significative à la fois la latence et la consommation de ressources matérielles. Ce chapitre a également détaillé l'implémentation des opérations arithmétiques de base—addition et multiplication—au sein du cadre CuFPSAF, en utilisant l'approche SAF pour optimiser l'alignement des données. Ces fonctions élémentaires constituent la base pour des opérations plus complexes telles que la somme vectorielle (`VSUM`), le produit scalaire (`DP`) et la multiplication matrice-vecteur (`MVM`). Pour les opérations vectorielles à haute performance comme le produit scalaire, CuFPSAF prend en charge deux stratégies d'implémentation distinctes : (i) Mode contraint en latence (LC, Latency-Constrained) : cette configuration traite l'ensemble

du vecteur en parallèle afin de minimiser la latence, au prix d'une consommation accrue de ressources matérielles. (ii) Mode contraint en ressources (RC, Resource-Constrained) : cette approche alternative divise le vecteur en segments plus petits traités séquentiellement, permettant la réutilisation de la logique arithmétique, réduisant ainsi considérablement la consommation de ressources matérielles, au prix d'une latence accrue.

Les évaluations expérimentales réalisées sur une FPGA Alveo U280 ont montré que CuFPSAF atteint des niveaux de performance et de précision comparables à ceux de la bibliothèque CuFP d'origine et des cœurs IP fournis par les fabricants. Dans les tâches de somme vectorielle, CuFPSAF a atteint la latence la plus faible et la meilleure utilisation des LUT parmi toutes les configurations évaluées. Pour un vecteur de taille 64, CuFPSAF a réduit le nombre de cycles d'exécution de 29,4% par rapport à CuFP et de 81,5% par rapport à l'IP fournisseur, tout en conservant une consommation de DSP similaire à celle de CuFP et en utilisant 59,7% de DSP en moins que l'IP fournisseur. Cette réduction est obtenue en passant de cinq (5) DSP par multiplieur FP plus additionneur à seulement deux (2), la différence de trois (3) provenant du fait que les IP Xilinx utilisent trois (3) DSP pour l'opération d'addition FP, ce qui n'est pas le cas dans l'implémentation proposée. Concernant les opérations de produit scalaire, CuFPSAF a systématiquement présenté la latence la plus faible parmi les variantes testées et a maintenu une efficacité matérielle compétitive, bien que l'utilisation des FF tende à augmenter avec la taille du vecteur. Pour la multiplication matrice-vecteur en mode RC, CuFPSAF a démontré une utilisation nettement plus faible des LUT, des FF et des DSP—en particulier pour les matrices de grande taille—tout en maintenant une latence comparable à celle des solutions CuFP et IP fournisseur. Dans l'ensemble, ce travail a démontré que l'intégration de la technique SAT dans CuFP a considérablement amélioré les capacités de la bibliothèque. Le cadre résultant, CuFPSAF, offre des améliorations notables en matière de latence, d'efficacité matérielle et de scalabilité pour les opérations en virgule flottante sur les systèmes à base de FPGA.

10.3.5 Article 5

Dans le Chapitre 9, nous nous sommes concentrés sur l'amélioration de l'efficacité des opérations d'addition sur FPGA, un composant essentiel pour l'accélération des calculs dans divers domaines tels que le traitement du signal, l'apprentissage automatique, et plus particulièrement, la simulation en temps réel. Étant donné que l'addition est l'une des opérations les plus fréquemment invoquées dans les moteurs de simulation en temps réel — en particulier dans les solveurs numériques impliquant des opérations matricielles, des sommes vectorielles et des algorithmes itératifs — la latence et l'efficacité matérielle de l'unité d'addition peuvent

avoir un impact significatif sur les performances globales du système. Le cœur de ce travail repose sur le format HRCS, conçu pour minimiser les délais de propagation de retenue en représentant chaque chiffre par une combinaison d'un mot de somme et d'un bit de retenue. Cette structure permet des opérations de sommation parallèles sans résolution immédiate des retenues, ce qui la rend particulièrement adaptée aux structures arborescentes de sommation. Ce chapitre a proposé une version améliorée de l'additionneur endomorphique HRCS à chaîne de retenue unique, nommée IESC-HRCS. Le circuit a été implémenté en VHDL et évalué à l'aide d'un cas test de réduction vectorielle, particulièrement pertinent pour les tâches de simulation en temps réel nécessitant un traitement à haut débit de grands vecteurs de données sous des contraintes temporelles strictes. Les résultats expérimentaux confirment que l'additionneur IESC-HRCS proposé permet des réductions de latence substantielles sans coût supplémentaire en ressources. Par exemple, dans le cadre d'une sommation vectorielle de 32 éléments (`vsum32`) avec une base (radix) de 64, l'additionneur amélioré a permis une réduction de latence de 29,5% par rapport à la version précédente, tout en conservant une utilisation comparable des LUTs. Ces résultats sont particulièrement prometteurs pour les applications de simulation en temps réel, où l'exécution déterministe et des latences inférieures à la microseconde sont requises. Les travaux futurs porteront sur l'intégration de l'IESC-HRCS dans des unités de traitement arithmétique complètes, ainsi que son extension à des opérations plus complexes telles que le produit scalaire (`dp`) et la multiplication matrice-vecteur. Ces opérations constituent en effet des éléments fondamentaux des solveurs en espace d'état utilisés dans les simulations EMT en temps réel sur plateformes FPGA.

10.3.6 Cohérence globale

Les cinq articles présentés dans cette thèse s'inscrivent dans une démarche de recherche structurée et progressive, construite autour de la problématique centrale de la simulation en temps réel sur FPGA. Chacun de ces travaux adresse un niveau spécifique de la chaîne de conception, depuis la modélisation des réseaux électriques jusqu'à l'implémentation matérielle optimisée d'opérateurs arithmétiques, en passant par la définition de représentations numériques adaptées.

L'ordre des publications reflète une logique ascendante :

— **L'article 1** pose les bases applicatives en démontrant la faisabilité de la simulation FDNE sur FPGA via HLS, tout en révélant les limites des représentations numériques classiques.

— **L'article 2** répond à ces limites par la proposition de la bibliothèque CuFP, qui permet une plus grande flexibilité et une meilleure maîtrise des compromis entre précision,

latence et ressources.

- **L'article 3** intègre cette bibliothèque dans un cadre complet de simulation d'un STATCOM, validant la robustesse et la pertinence de l'approche dans un contexte industriel réaliste.
- **L'article 4** approfondit l'optimisation des performances arithmétiques en introduisant une version améliorée de CuFP (CuFPSAF), qui exploite le SAT pour alléger les opérations critiques dans les chemins de données.
- **Enfin, l'article 5** cible spécifiquement l'opération d'addition — fréquemment utilisée dans les solveurs de simulation — en proposant une architecture endomorphique performante (IESC-HRCS), ouvrant ainsi la voie à la création d'unités arithmétiques spécialisées à très faible latence. Bien que cet article ne soit pas complètement aligné avec le reste de la thèse, il apporte néanmoins une contribution pertinente à l'amélioration des performances des opérations d'addition, susceptibles de bénéficier aux applications visées. Ce travail n'a malheureusement pas été réalisé dans le cadre de cette thèse, mais il serait relativement facile à reproduire.

L'enchaînement de ces contributions témoigne d'une cohérence méthodologique, où chaque développement repose sur les acquis des précédents. Ensemble, ils forment un corpus solide qui répond à la problématique initiale en proposant à la fois des outils pratiques (bibliothèques HLS), des méthodologies de modélisation, et des innovations architecturales pour la simulation temps réel sur FPGA. Cette continuité renforce la valeur scientifique de l'ensemble de la thèse et ouvre des perspectives concrètes pour la conception de simulateurs performants, flexibles et accessibles aux non-spécialistes du matériel.

CHAPITRE 11 CONCLUSION

Dans cette thèse, une méthodologie pour le développement de simulateurs en temps réel sur des plateformes FPGA a été proposée. Cette méthodologie a présenté un cadre de conception structuré et de haut niveau, spécifiquement dédié à la simulation en temps réel de FDNE. L'objectif principal était de combler le fossé entre la complexité computationnelle liée à la modélisation des réseaux électriques et les exigences strictes d'exécution en temps réel, notamment dans les environnements HIL. Pour ce faire, les outils de synthèse de haut niveau ont été exploités afin de faciliter le développement matériel pour les ingénieurs applicatifs, tout en garantissant des performances comparables à celles des conceptions réalisées en RTL.

Tout au long des chapitres, ce travail a présenté plusieurs contributions majeures. Il a proposé diverses approches pour la modélisation de FDNE et les a mises en œuvre sur des plateformes FPGA en utilisant des formulations en espace d'état. L'étude a examiné en profondeur l'impact de la résolution numérique sur la simulation en temps réel, en évaluant les performances de différents types de données, y compris les formats à virgule flottante standard (simple et double précision), les formats à virgule fixe et les formats personnalisés. Pour améliorer davantage les performances et la flexibilité de conception, deux bibliothèques dédiées à la synthèse de haut niveau ont été développées : CuFP et CuFPSAF. Ces bibliothèques permettent la configuration personnalisée des formats flottants ainsi qu'un alignement efficace des exposants. Elles ont montré des améliorations notables : CuFPSAF a permis une réduction de latence de 29,4% par rapport à CuFP et de plus de 80% par rapport aux blocs IP du fournisseur pour une sommation vectorielle de 64 éléments, tout en réduisant l'utilisation des ressources DSP de 59,7%. De plus, une architecture d'additionneur améliorée, appelée IESC-HRCS, a été introduite pour optimiser les performances dans les opérations intensives en additions. Pour une sommation vectorielle de 32 éléments utilisant un format de base 64, l'IESC-HRCS a réduit la latence de 29,5% par rapport aux conceptions antérieures à chaîne unique, tout en conservant une utilisation comparable des ressources logiques. Ces contributions ont été validées à travers des études de cas portant sur des systèmes électriques embarqués dans l'aéronautique et des convertisseurs connectés au réseau, et ont démontré une exécution inférieure à la microseconde, une scalabilité vers des modèles de plus grande taille, et une précision numérique élevée, avec des erreurs relatives en norme 2 aussi faibles que 0,0103% pour la double précision et 0,0868% pour CuFP(8,24).

La principale limitation de cette méthodologie réside dans l'absence de reconfigurabilité de l'implémentation. En effet, la conception actuelle ne propose ni surcouche (overlay) ni cadre

matériel flexible, ce qui implique qu'une modification du modèle FDNE ou de son interaction avec d'autres composants — comme un changement de cas d'étude ou de paramètres du système — nécessite une nouvelle synthèse complète et un redéploiement sur la plateforme FPGA. Une autre limitation concerne la simplicité du cas d'étude choisi, qui se limite à un seul convertisseur connecté au réseau. Néanmoins, cette configuration permet de démontrer la faisabilité de la méthodologie proposée et met en évidence son potentiel à réaliser des simulations EMT à haute résolution avec une très faible latence. De plus, elle illustre la flexibilité en matière de gestion de la précision offerte par CuFP, ouvrant ainsi la voie à des extensions futures vers des scénarios de réseaux électriques plus complexes.

En s'appuyant sur les résultats et les limitations de ce mémoire, plusieurs améliorations peuvent être apportées afin de généraliser cette méthodologie et de l'adapter à des études et développements futurs.

Premièrement, pour surmonter la limitation liée au manque de reconfigurabilité, les travaux futurs devraient se concentrer sur le développement d'une architecture à surcouche (overlay) dynamique et modulaire. Une telle approche permettrait la réutilisation des composants de calcul principaux et l'adaptation rapide à de nouveaux modèles FDNE ou à différents cas d'étude, sans nécessiter une synthèse complète et un redéploiement sur la plateforme FPGA.

Deuxièmement, la méthodologie devrait être validée dans des scénarios de réseaux électriques plus complexes, incluant notamment des réseaux à grande échelle et des systèmes multi-convertisseurs. Cela permettrait d'évaluer la scalabilité et la robustesse des bibliothèques et des architectures proposées dans des contextes opérationnels variés.

RÉFÉRENCES

[1] T. Ould-Bachir et J. P. David, "Self-alignment schemes for the implementation of addition-related floating-point operators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 6, nº. 1, May 2013.

[2] T. Ould-Bachir, "Opérateurs et engins de calcul en virgule flottante et leur application à la simulation en temps réel sur FPGA," Thèse de doctorat, École Polytechnique de Montréal, August 2013.

[3] F. Hajizadeh, L. Alavoine, T. Ould-Bachir, F. Sirois et J. P. David, "FPGA-based FDNE models for the accurate real-time simulation of power systems in aircrafts," dans *International Conference on Renewable Energy Research and Applications (ICRERA)*, 2023, p. 344–348.

[4] F. Hajizadeh, T. Ould-Bachir et J. P. David, "CuFP : An HLS library for customized floating-point operators," *Electronics*, vol. 13, nº. 14, 2024.

[5] F. Hajizadeh, A. Masoom, T. Ould-Bachir et J. P. David, "FPGA-based simulation of grid-tied converters using frequency-dependent network equivalent," *16th edition of the International Conference on Power Systems Transients*, 2024.

[6] F. Hajizadeh, T. Ould-Bachir et J. P. David, "Enhancing CuFP library with self-alignment technique," *Computers*, vol. 14, nº. 4, 2025.

[7] F. Hajizadeh, P.-A. Bisson, T. Ould-Bachir et J. P. David, "Improved high-radix carry-save adders for low-latency vector reduction on FPGA," *23rd IEEE International NEWCAS Conference*, p. 1–5, 2025.

[8] F. Hajizadeh, L. Alavoine, T. Ould-Bachir, F. Sirois et J. P. David, "FPGA-based FDNE models for the accurate real-time simulation of power systems in aircrafts," dans *2023 12th International Conference on Renewable Energy Research and Applications (ICRERA)*, 2023, p. 344–348.

[9] M. Sotero, G. Fontenele, F. Dicler *et al.*, "An FPGA-based hardware-in-the-loop implementation of power electronics circuits using a generic real-time simulator," dans *Brazilian Power Electronics Conference (COBEP)*, 2021, Conference Proceedings, p. 1–8.

[10] L. Gaitan, J. Sanchez et L. Velazquez, "A review of real time digital simulations : Theory and applications for the energy transition," *IEEE Latin America Transactions*, vol. 20, nº. 10, p. 2295–2307, 2022.

[11] X. Guillaud, M. O. Faruque, A. Teninge *et al.*, "Applications of real-time simulation technologies in power and energy systems," *IEEE Power and Energy Technology Systems Journal*, vol. 2, nº. 3, p. 103–115, 2015.

[12] J. Bélanger, P. Venne et J. N. Paquin, "The what, where, and why of real-time simulation," *Planet RT*, p. 37–49, 2010.

[13] X. Zheng, "Real-time simulation in real-time systems : Current status, research challenges and a way forward," *arXiv [cs.DC]*, 2019. [En ligne]. Disponible : http://arxiv.org/abs/1905.01848

[14] F. Montano, "Architectures and methodology for the design of real-time power converter simulators on FPGAs," Ph.D. thesis, 2021.

[15] G. F. Lauss, M. O. Faruque, K. Schoder *et al.*, "Characteristics and design of power hardware-in-the-loop simulations for electrical power systems," *IEEE Transactions on Industrial Electronics*, vol. 63, nº. 1, p. 406–417, 2016.

[16] B. Jandaghi et V. R. Dinavahi, "Real-time HIL emulation of faulted electric machines based on nonlinear MEC model," *IEEE Transactions on Energy Conversion*, vol. 34, p. 1190–1199, 2019.

[17] M. S. R. Leal, L. D. Simões, R. L. S. França *et al.*, "Methodology to perform real-time simulation of power systems using a FPGA-based platform," dans *Workshop on Communication Networks and Power Systems (WCNPS)*, 2018, p. 1–5.

[18] K. Meddah, H. Chalangar et T. Ould-Bachir, "Real-time simulation of a fast charger using a low-cost FPGA platform," dans *IECON 48th Annual Conference of the IEEE Industrial Electronics Society*, 2022, Conference Proceedings, p. 1–6.

[19] Y. Chen et V. Dinavahi, "FPGA-based real-time EMTP," *IEEE Transactions on Power Delivery*, vol. 24, nº. 2, p. 892–902, 2009.

[20] I. Bahri, M. W. Naouar, E. Monmasson, I. Slama-Belkhodja et L. Charaabi, "Design of an FPGA-based real-time simulator for electrical system," *2008 13th International Power Electronics and Motion Control Conference*, p. 1365–1370, 2008.

[21] C. Yang, Y. Xue et X. P. Zhang, "FPGA-based detailed EMTP," dans *Proceedings of the IEEE Manchester PowerTech*, 2017, p. 1–6.

[22] S. Subedi, M. Rauniyar, S. Ishaq *et al.*, "Review of methods to accelerate electromagnetic transient simulation of power systems," *IEEE Access*, vol. 9, p. 89 714–89 731, 2021.

[23] Y. F. Oliveira, F. A. La-Gatta, R. A. F. Ferreira et M. C. B. P. Rodrigues, "Development of a FPGA-based real-time simulation system," dans *IEEE 15th Brazilian*

Power Electronics Conference and 5th IEEE Southern Power Electronics Conference (COBEP/SPEC), 2019, Conference Proceedings, p. 1–6.

[24] S. Lahti, P. Sjövall, J. Vanne et T. D. Hämäläinen, "Are we there yet ? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, n°. 5, p. 898–911, 2019.

[25] F. Montaño, T. Ould-Bachir et J. P. David, "A latency-insensitive design approach to programmable FPGA-based real-time simulators," *Electronics*, vol. 9, n°. 11, 2020.

[26] J. Morales Rodriguez, E. Medina, J. Mahseredjian *et al.*, "Frequency-domain fitting techniques : A review," *IEEE Transactions on Power Delivery*, vol. 35, n°. 3, p. 1102–1110, 2020.

[27] L. Xie, Y. Xiang, Q. Mu et Q. Mu, "The hardware implementation of FPGA-based electromagnetic transient system for new energy components," dans *IEEE/PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, 2019, Conference Proceedings, p. 1–5.

[28] B. C. Schafer et Z. Wang, "High-level synthesis design space exploration : Past, present, and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, n°. 10, p. 2628–2639, 2020.

[29] J. Cong, J. Lau, G. Liu *et al.*, "FPGA HLS today : Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, n°. 4, p. Article 51, 2022.

[30] S. Lahti, M. Rintala et T. D. Hämäläinen, "Leveraging modern C++ in high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, n°. 4, p. 1123–1132, 2023.

[31] R. Campos et J. M. P. Cardoso, "On data parallelism code restructuring for HLS targeting FPGAs," dans *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, Conference Proceedings, p. 144–151.

[32] L. Huang, D. Li, K. Wang, T. Gao et A. Tavares, "A survey on performance optimization of high-level synthesis tools," *Journal of Computer Science and Technology*, vol. 35, n°. 3, p. 697–720, 2020.

[33] M. Siracusa, E. D. Sozzo, M. Rabozzi *et al.*, "A comprehensive methodology to optimize FPGA designs via the roofline model," *IEEE Transactions on Computers*, vol. 71, n°. 8, p. 1903–1915, 2022.

[34] R. S. Molina, V. Gil-Costa, M. L. Crespo et G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators : Models, methodologies, and frameworks," *IEEE Access*, vol. 10, p. 90 429–90 455, 2022.

[35] F. Montano, T. Ould-Bachir et J. P. David, “An evaluation of a high-level synthesis approach to the FPGA-based submicrosecond real-time simulation of power converters,” *IEEE Transactions on Industrial Electronics*, vol. 65, n°. 1, p. 636–644, 2018.

[36] XilinxRTLExport. www.xilinx.com/rtlexport. [En ligne]. Disponible : https://docs.xilinx.com/r/en-US/ug1399-vitis-hls/Fixed-Point-Identifier-Summary

[37] “IEEE standard for floating-point arithmetic,” *IEEE Std 754-2008*, p. 1–70, 2008.

[38] M. Martel, “Enhancing the implementation of mathematical formulas for fixed-point and floating-point arithmetics,” *Formal Methods in System Design*, vol. 35, p. 265–278, 12 2009.

[39] A. Sanchez, E. Todorovich et A. De Castro, “Exploring the limits of floating-point resolution for hardware-in-the-loop implemented with FPGAs,” *Electronics*, vol. 7, n°. 10, 2018.

[40] M. S. Martínez-García, A. de Castro, A. Sanchez et J. Garrido, “Analysis of resolution in feedback signals for hardware-in-the-loop models of power converters,” *Electronics*, vol. 8, n°. 12, 2019.

[41] X. Wang et M. Leeser, “VFloat : A variable precision fixed- and floating-point library for reconfigurable hardware,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, n°. 3, sep 2010.

[42] F. de Dinechin et B. Pasca, “Designing custom arithmetic data paths with FloPoCo,” *IEEE Design Test of Computers*, vol. 28, n°. 4, p. 18–27, 2011.

[43] F. de Dinechin, “Reflections on 10 years of FloPoCo,” dans *ARITH 2019 - 26th IEEE Symposium on Computer Arithmetic*, kyoto, Japan, juin 2019, p. 1–3. [En ligne]. Disponible : https://inria.hal.science/hal-02161527

[44] S. Bansal, H. Hsiao, T. Czajkowski et J. H. Anderson, “High-level synthesis of software-customizable floating-point cores,” dans *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, p. 37–42.

[45] Y. Uguen, F. Dinechin et S. Derrien, “Bridging high-level synthesis and application-specific arithmetic : The case study of floating-point summations,” dans *27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, Conference Proceedings, p. 1–8.

[46] D. B. Thomas, “Templatised soft floating-point for high-level synthesis,” dans *IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2019, p. 227–235.

[47] D. B. Thomas, “Compile-time generation of custom-precision floating-point IP using HLS tools,” dans *IEEE Symposium on Computer Arithmetic (ARITH)*, 2019, p. 192–

193.

[48] M. Fiorito, S. Curzel et F. Ferrandi, “TrueFloat : A templatized arithmetic library for HLS floating-point operators,” dans *Embedded Computer Systems : Architectures, Modeling, and Simulation*, C. Silvano, C. Pilato et M. Reichenbach, édit., 2023, p. 486–493.

[49] F. Ferrandi, V. G. Castellana, S. Curzel *et al.*, “Invited : Bambu : an open-source research framework for the high-level synthesis of complex applications,” dans *ACM/IEEE Design Automation Conference (DAC)*, 2021, p. 1327–1330.

[50] D. Filippas, C. Nicopoulos et G. Dimitrakopoulos, “Templatized fused vector floating-point dot product for high-level synthesis,” *Journal of Low Power Electronics and Applications*, vol. 12, n°. 4, 2022.

[51] X. Zhou, G. He et X. Zhou, “FPGA design and implementation for real-time electromagnetic transient simulation system,” dans *IEEE International Conference on High Performance Computing and Communications*, 2015, Conference Proceedings, p. 848–851.

[52] Y. Chen et V. Dinavahi, “FPGA-based real-time EMTP,” *IEEE Transactions on Power Delivery*, vol. 24, n°. 2, p. 892–902, 2009.

[53] M. Matar et R. Iravani, “The reconfigurable-hardware real-time and faster-than-real-time simulator for the analysis of electromagnetic transients in power systems,” *IEEE Transactions on Power Delivery*, vol. 28, n°. 2, p. 619–627, 2013.

[54] L. F. C. de Sá Santos Ribeiro, F. N. F. Dicler, L. G. B. Rolim et M. Aredes, “Real-time implementation of a DC converter using modified nodal analysis, sparsity handling and parallelism on a DSP platform,” dans *IEEE Brazilian Power Electronics Conference and IEEE Southern Power Electronics Conference (COBEP/SPEC)*, 2019, Conference Proceedings, p. 1–6.

[55] H. Chalangar, T. Ould-Bachir, K. Sheshyekani et J. Mahseredjian, “Methods for the accurate real-time simulation of high-frequency power converters,” *IEEE Transactions on Industrial Electronics*, vol. 69, n°. 9, p. 9613–9623, 2022.

[56] R. Sriganesh, M. Pandikumar et R. Sundareswaran, “FPGA based real time simulation of power converters,” dans *Proceedings of the International Conference on Electrical Energy Systems (ICEES)*, 2021, p. 370–376.

[57] R. Iracheta-Cortez, N. Flores-Guzman et R. Hasimoto-Beltran, “Implementation of the frequency dependent line model in a real-time power system simulator,” *Ingeniería e Investigación*, vol. 37, n°. 3, p. 61–71, 2017.

[58] T. Ould-Bachir, H. Chalangar, K. Sheshyekani et J. Mahseredjian, “High performance computing engines for the FPGA-based simulation of the ULM,” *Electric Power Systems Research*, vol. 190, 2021.

[59] Q. Li, Y. Xiang, Q. Mu *et al.*, “Exploration of FPGA-based electromagnetic transient real-time simulation system design using high-level synthesis,” *The Journal of Engineering*, vol. 2019, n°. 16, p. 1217–1220, 2019.

[60] F. Dicler, “Contributions to the FPGA and CPU implementation of frequency-dependent network equivalents for real-time and offline electromagnetic transient power system simulators,” Thesis, 2021.

[61] M. Rivard, C. Fallaha et J.-N. Paquin, “Real-time simulation of a more electric aircraft power generation and distribution system,” dans *AIAA/IEEE Electric Aircraft Technologies Symposium (EATS)*, 2018, p. 1–10.

[62] A. S. Vijay, S. Doolla et M. C. Chandorkar, “Real-time testing approaches for microgrids,” *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 5, n°. 3, p. 1356–1376, 2017.

[63] J. Chen, C. Wang et J. Chen, “Investigation on the selection of electric power system architecture for future more electric aircraft,” *IEEE Transactions on Transportation Electrification*, vol. 4, n°. 2, p. 563–576, 2018.

[64] L. M. Lobo, C. Dufour et J. Mahseredjian, “Real-time simulation of more-electric aircraft power systems,” dans *15th European Conference on Power Electronics and Applications (EPE)*, 2013, p. 1–10.

[65] T. Ould-Bachir, H. Chalangar, K. Sheshyekani et J. Mahseredjian, “High performance computing engines for the FPGA-based simulation of the ULM,” *Electric Power Systems Research*, vol. 190, 2021.

[66] M. Berljafa et S. Güttel, “Generalized rational Krylov decompositions with an application to rational approximation,” *SIAM Journal on Matrix Analysis and Applications*, vol. 36, n°. 2, p. 894–916, 2015.

[67] Y. Uguen, F. D. Dinechin, V. Lezaud et S. Derrien, “Application-specific arithmetic in high-level synthesis tools,” *ACM Trans. Archit. Code Optim.*, vol. 17, n°. 1, mar 2020.

[68] Xilinx, “UG902 : Vivado design suite user guide,” 2021.

[69] Xilinx, “PG060 : Logicore IP product guide,” 2020.

[70] Xilinx, “UG900 : Vivado design suite user guide : Logic simulation,” 2024.

[71] Intel, “Floating-point IP cores user guide,” 2023.

[72] S. Cherubin, D. Cattaneo, M. Chiari, A. D. Bello et G. Agosta, "Taffo : Tuning assistant for floating to fixed point optimization," *IEEE Embedded Systems Letters*, vol. 12, nº. 1, p. 5–8, 2020.

[73] G. Agosta, "Precision tuning of mathematically intensive programs : a comparison study between fixed point and floating point representations," Mémoire de maîtrise, School of Industrial and Information Engineering, Politecnico di Milano, 2021.

[74] D. Cattaneo, M. Chiari, N. Fossati, S. Cherubin et G. Agosta, "Architecture-aware precision tuning with multiple number representation systems," dans *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, p. 673–678.

[75] M. Langhammer et T. VanCourt, "FPGA floating point datapath compiler," dans *IEEE Symposium on Field Programmable Custom Computing Machines*, 2009, p. 259–262.

[76] Xilinx, "UG579 : Ultrascale architecture dsp slice," 2021.

[77] X. Fang et M. Leeser, "Open-source variable-precision floating-point library for major commercial fpgas," vol. 9, nº. 3, jul 2016.

[78] K. M. de Dinechin, Florent, *Application-Specific Arithmetic*. Springer International Publishing, 2023.

[79] A. Böttcher, M. Kumm et F. de Dinechin, "Resource optimal truncated multipliers for FPGAs," dans *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*, 2021, p. 102–109.

[80] A. Böttcher et M. Kumm, "Towards globally optimal design of multipliers for FPGAs," *IEEE Transactions on Computers*, vol. 72, nº. 5, p. 1261–1273, 2023.

[81] F. Montaño, T. Ould-Bachir et J. P. David, "A latency-insensitive design approach to programmable FPGA-based real-time simulators," *Electronics*, vol. 9, nº. 11, 2020.

[82] A. Perera, R. Nilsen, T. Haugan et K. Ljokelsoy, "A design method of an embedded real-time simulator for electric drives using low-cost system-on-chip platform," dans *Proceedings of the International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*, 2021, p. 1–8.

[83] E. Zamiri, A. Sanchez, M. Yushkova, M. S. Martínez-García et A. de Castro, "Comparison of different design alternatives for hardware-in-the-loop of power converters," *Electronics*, nº. 8, 2021.

[84] T. Ould-Bachir et J.-P. David, "Performing floating-point accumulation on a modern fpga in single and double precision," dans *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, p. 105–108.

[85] B. Parhami, *Computer arithmetic : Algorithms and hardware designs*, ser. Oxford series in electrical and computer engineering. Oxford University Press, 2010.

[86] Xilinx, “UG1399 : Vitis high-level synthesis user guide,” 2023.

[87] N. Burham, “Study of STATCOM for voltage compensation in 14-bus IEEE grid,” Thesis, Uppsala University, 2024.

[88] C. K. Tse, M. Huang, X. Zhang, D. Liu et X. L. Li, “Circuits and systems issues in power electronics penetrated power grid,” *IEEE Open Journal of Circuits and Systems*, p. 140–156, 2020.

[89] S. Kouro, J. I. Leon, D. Vinnikov et L. G. Franquelo, “Grid-connected photovoltaic systems : An overview of recent research and emerging PV converter technology,” *IEEE Industrial Electronics Magazine*, vol. 9, n°. 1, p. 47–61, 2015.

[90] S. Rivera, S. Kouro, S. Vazquez *et al.*, “Electric vehicle charging infrastructure : From grid to battery,” *IEEE Industrial Electronics Magazine*, vol. 15, n°. 2, p. 37–51, 2021.

[91] N. Kumar, P. Wagh, D. Kolhe, P. Arane et P. Kadlag, “Power quality improvement in distributed energy resources for EV charging using STATCOM,” dans *Proceedings of the International Conference on Sustainable Technology for Power and Energy Systems (STPES)*, 2022, p. 1–4.

[92] R. Sahoo et M. Roy, “An FPGA-based balancing of capacitor voltage for a five-level CHB inverter,” *Arabian Journal for Science and Engineering*, vol. 49, 04 2024.

[93] C.-Y. Tang et J.-H. Jheng, “An active power ripple mitigation strategy for three-phase grid-tied inverters under unbalanced grid voltages,” *IEEE Transactions on Power Electronics*, vol. 38, n°. 1, p. 27–33, 2023.

[94] Y. Chen, B. Zhang, D. Qiu *et al.*, “Switched active power control of a grid-connected inverter with reduced rocof and frequency overshoot,” *IEEE Transactions on Power Electronics*, vol. 39, n°. 4, p. 4062–4077, 2024.

[95] A. P. Murdan, I. Jahmeerbacus et S. Z. S. Hassen, “Modeling and simulation of a STATCOM for reactive power control,” dans *Proceedings of the International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM)*, 2022, p. 1–6.

[96] M. D. Omar Faruque, T. Strasser, G. Lauss *et al.*, “Real-time simulation technologies for power systems design, testing, and analysis,” *IEEE Power and Energy Technology Systems Journal*, vol. 2, n°. 2, p. 63–73, 2015.

[97] B. Sullivan, J. Shi, M. Mazzola et B. Saravi, “Faster-than-real-time power system transient stability simulation using parallel general norton with multiport equivalent (PGNME),” dans *IEEE Power Energy Society General Meeting (PESGM)*, 2017, p. 1–5.

[98] X. Liu, J. Ospina, I. Zografopoulos, A. Russel et C. Konstantinou, “Faster than real-time simulation : methods, tools, and applications,” dans *Proceedings of the 9th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*, ser. MSCPES ’21. New York, NY, USA : Association for Computing Machinery, 2021.

[99] X. Guillaud, M. O. Faruque, A. Teninge *et al.*, “Applications of real-time simulation technologies in power and energy systems,” *IEEE Power and Energy Technology Systems Journal*, vol. 2, n°. 3, p. 103–115, 2015.

[100] X. Lin, A. M. Gole et M. Yu, “A wide-band multi-port system equivalent for real-time digital power system simulators,” *IEEE Transactions on Power Systems*, vol. 24, n°. 1, p. 237–249, 2009.

[101] J. Morales, J. Mahseredjian, A. Ramirez, K. Sheshyekani et I. Kocar, “A Loewner/MPM—VF combined rational fitting approach,” *IEEE Transactions on Power Delivery*, vol. 35, n°. 2, p. 802–808, 2019.

[102] K. Xie, Q. Lu, H. Jiang et H. Wang, “Accurate sum and dot product with new instruction for high-precision computing on ARMv8 processor,” *Mathematics*, vol. 13, n°. 2, 2025.

[103] J. Sohn et E. E. Swartzlander, “Improved architectures for a floating-point fused dot product unit,” dans *IEEE Symposium on Computer Arithmetic*, 2013, p. 41–48.

[104] E. Jamro, A. Dąbrowska-Boruch, P. Russek, M. Wielgosz et K. Wiatr, “Novel architecture for floating point accumulator with cancelation error detection,” *Bulletin of the Polish Academy of Sciences Technical Sciences*, 11 2023.

[105] J. Gao, J. Shen, Y. Zhang, W. Ji et H. Huang, “Precision-aware iterative algorithms based on group-shared exponents of floating-point numbers,” 2024.

[106] X. Wang et M. Leeser, “VFloat : A variable precision fixed- and floating-point library for reconfigurable hardware,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, n°. 3, sept. 2010.

[107] E. E. Swartzlander et H. H. Saleh, “FFT implementation with fused floating-point operations,” *IEEE Transactions on Computers*, vol. 61, n°. 2, p. 284–288, 2012.

[108] R. Nane, V. M. Sima, C. Pilato *et al.*, “A survey and evaluation of FPGA high-level synthesis tools,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, n°. 10, p. 1591–1604, 2016.

[109] Z. Luo et M. Martonosi, “Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques,” *IEEE Transactions on Computers*, vol. 49, n°. 3, p. 208–218, 2000.

[110] S. Vangal, Y. Hoskote, N. Borkar et A. Alvandpour, “A 6.2-GFlops floating-point multiply-accumulator with conditional normalization,” *IEEE Journal of Solid-State Circuits*, vol. 41, n°. 10, p. 2314–2323, 2006.

[111] K. Compton et S. Hauck, “Reconfigurable computing : A survey of systems and software,” *ACM Comput. Surv.*, vol. 34, n°. 2, p. 171–210, juin 2002.

[112] Xilinx, “Adder/subtracter v12.0 LogiCORE IP product guide,” https://https://docs.amd.com/v/u/en-US/pg120-c-addsub, Feb. 2021, pG120.

[113] D. L. N. Hettiarachchi, V. S. P. Davuluru et E. J. Balster, “Integer vs. floating-point processing on modern FPGA technology,” dans *Proceedings of the Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, p. 0606–0612.

[114] W. Hwu et S. Patel, “Accelerator architectures —a ten-year retrospective,” *IEEE Micro*, vol. 38, n°. 6, p. 56–62, 2018.

[115] P. Gorlani, T. Kenter et C. Plessl, “OpenCL implementation of cannon’s matrix multiplication algorithm on Intel Stratix 10 FPGAs,” dans *Proceedings of the International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, p. 99–107.

[116] Xilinx, “Ultrascale+ FPGAs product tables and product selection guide,” https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf, Dec 2020, last accessed March 2022.

[117] Intel, “Intel FPGA product catalog v21.3,” https://www.intel.com/content/dam/www/central-libraries/us/en/documents/product-catalog.pdf, 2021, last accessed March 2022.

[118] A. Boutros et V. Betz, “FPGA architecture : Principles and progression,” *IEEE Circuits and Systems Magazine*, vol. 21, n°. 2, p. 4–29, 2021.

[119] Ananthakrishnan, A. Ajit, A. V *et al.*, “FPGA based performance comparison of different basic adder topologies with parallel processing adder,” dans *Proceedings of the International conference on Electronics, Communication and Aerospace Technology (ICECA)*, 06 2019, p. 87–92.

[120] M. Jayabalan, S. M et P. A, “Performance comparison of adder topologies with parallel processing adder circuit,” *Innovations in Information and Communication Technology Series*, p. 209–211, 12 2020.

[121] C. Liu, F. Qiao, X. Yang et H. Yang, “Hardware acceleration with pipelined adder for support vector machine classifier,” *Proceedings of the International Conference on Digital Information and Communication Technology and its Applications (DICTAP)*, p. 13–16, 2014.

[122] H. Li, Z. Liang, H. Li et Y. Ye, “A high-performance wide FPGA adder based on carry chains,” ser. EITCE ’20. New York, NY, USA : Association for Computing Machinery, 2021, p. 860–864.

[123] R. Dornelles, G. Paim, B. Silveira *et al.*, “A power-efficient 4-2 adder compressor topology,” dans *2017 15th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2017, p. 281–284.

[124] M. U. Haque, Z. T. Sworna et H. Babu, “A fast FPGA-based BCD adder,” *Circuits, Systems, and Signal Processing*, vol. 37, 10 2018.

[125] P. Ienne, “Introduction to the special section on FPGA 2022,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, nº. 4, déc. 2023.

[126] S. Velagaleti et G. Hruthik, “Area-efficient fault detection mechanism for carry-lookahead and carry-save adders,” dans *Proceedings of the Asia Pacific Conference on Innovation in Technology (APCIT)*, 2024, p. 1–6.